হাবিজাবি হিজিবিজবিজ

yet another হ জ ব র ল programming blog

- পরথম পাতা
- আমি
- কোড লাইব্রেরি
- অন্যান্য লিঙ্ক
-

Type text to search here...

কোড লাইব্রেরি

আমার কনটেস্ট কোড লাইব্রেরি – কমন কিছু অ্যালগরিদমের স্কেলিটন কোড। অ্যালগরিদম সম্পর্কে কারো পর্যাপ্ত ধারণা না থাকলে তাকে এই কোড লাইব্রেরি এর কোড কপি করার ক্ষেত্রে নিরুৎসাহিত করা যাচ্ছে, কারন সেক্ষেত্রে অধিকাংশ কোডই ঠিক মত কাজ করবে না। কোড গুলোর উপরে কমেন্ট সেকশন গুরুত্ব দিয়ে পড়ার জন্যও অনুরোধ থাকল, পরে প্রবলেমে পেন্যাল্টি খেয়ে আবার বলতে আসবেন না যে আমার এই কোড ভুয়া। এগুলা সবই অনেকবার টেস্ট করা কোড (টাইপিং মিসটেক থাকলে সেটা অন্য কথা, যদিও থাকার চ্যান্স খুবই কম), ধন্যবাদ।

ইনডেক্সঃ

কন্টেস্ট টেমপ্লেট [ইনডেক্স]

```
1   /*
2   USER: zobayer
3   TASK:
4   ALGO:
5   */
6
7   //#pragma warning (disable: 4786)
8   //#pragma comment (linker, "/STACK:0x800000")
9   //#define _CRT_SECURE_NO_WARNINGS 1
10
11  #include <cassert>
12  #include <cctype>
13  #include <climits>
14  #include <cmath>
15  #include <cstdio>
16  #include <cstdlib>
17  #include <cstring>
18  #include <iostream>
19  #include <sstream>
20  #include <iomanip>
21  #include <string>
22  #include <vector>
23  #include <list>
24  #include <set>
25  #include <map>
26  #include <stack>
27  #include <queue>
28  #include <algorithm>
29  #include <iterator>
30  #include <utility>
31  using namespace std;
32
33  template< class T > T _abs(T n) { return (n < 0 ? -n : n); }
34  template< class T > T _max(T a, T b) { return (!(a < b) ? a
35  template< class T > T _min(T a, T b) { return (a < b ? a : b
36  template< class T > T sq(T x) { return x * x; }
```

```
37
38    #define ALL(p) p.begin(),p.end()
39    #define MP(x, y) make_pair(x, y)
40    #define SET(p) memset(p, -1, sizeof(p))
41    #define CLR(p) memset(p, 0, sizeof(p))
42    #define MEM(p, v) memset(p, v, sizeof(p))
43    #define CPY(d, s) memcpy(d, s, sizeof(s))
44    #define READ(f) freopen(f, "r", stdin)
45    #define WRITE(f) freopen(f, "w", stdout)
46    #define SZ(c) (int)c.size()
47    #define PB(x) push_back(x)
48    #define ff first
49    #define ss second
50    #define i64 long long
51    #define ld long double
52    #define pii pair< int, int >
53    #define psi pair< string, int >
54
55    const double EPS = 1e-9;
56    const int INF = 0x7f7f7f7f;
57
58    int main() {
59        //READ("in.txt");
60        //WRITE("out.txt");
61        return 0;
62    }
```

ফাস্ট রিডার [ইনডেক্স]

```
1     /*
2     In gcc/g++ fread_unlocked() is even faster.
3     Can be made faster in non-object-oriented fashion with macro
4     In contest, who cares dynamic allocation?
5     */
6
7     struct FastRead {
8         char *buff, *ptr;
9         FastRead(int size) {
10            buff = new char[size];
11            ptr = buff;
12            fread(buff, 1, size, stdin);
13        }
14        ~FastRead() {
15            delete[] buff;
16        }
17        int nextInt() {
18            int ret = 0;
19            while(*ptr < '0' || *ptr > '9') ptr++;
20            do { ret = ret * 10 + *ptr++ - '0';
21            } while(*ptr >= '0' && *ptr <= '9');
22            return ret;
23        }
24    };
```

টাইম ট্র্যাকার [ইনডেক্স]

```
1     /*
```

```
 2    Usage: Just create an instance of it at the start of the blc
 3    which you want to measure running time. Reliable up to milli
 4    scale. You don't need to do anything else, even works with f
 5    */
 6
 7    class TimeTracker {
 8        clock_t start, end;
 9    public:
10        TimeTracker() {
11            start = clock();
12        }
13        ~TimeTracker() {
14            end = clock();
15            fprintf(stderr, "%.3lf s\n", (double)(end - start) /
16        }
17    };
```

পপ কাউন্ট [ইনডেক্স]

```
 1    /*
 2    Counts number of set bits in an integer.
 3    For 32 bit integer, delete 8 hex from const values and remov
 4    */
 5
 6    typedef __int64 INT;
 7
 8    const INT b1  = 0x5555555555555555;
 9    const INT b2  = 0x3333333333333333;
10    const INT b4  = 0x0f0f0f0f0f0f0f0f;
11    const INT b8  = 0x00ff00ff00ff00ff;
12    const INT b16 = 0x0000ffff0000ffff;
13    const INT b32 = 0x00000000ffffffff;
14
15    int popcount(INT x) {
16        x = (x & b1 ) + (( x >>  1 ) & b1 ); // 2 bits
17        x = (x & b2 ) + (( x >>  2 ) & b2 ); // 4 bits
18        x = (x & b4 ) + (( x >>  4 ) & b4 ); // 8 bits
19        x = (x & b8 ) + (( x >>  8 ) & b8 ); //16 bits
20        x = (x & b16) + (( x >> 16 ) & b16); //32 bits
21        x = (x & b32) + (( x >> 32 ) & b32); //64 bits
22        return x;
23    }
```

হিস্টোগ্রাম প্রবলেম [ইনডেক্স]

```
 1    /*
 2    Finds largest rectangular area in a histogram in O(n)
 3    */
 4
 5    i64 calc(int *ht, int n) {
 6        i64 ret = 0;
 7        int top = 1, st[MAX], i;
 8        ht[0] = st[0] = ht[++n] = 0;
 9        for(i = 1; i <= n; i++) {
10            while(top > 1 && ht[st[top-1]] >= ht[i]) {
11                ret = _max(ret, (i64)ht[st[top-1]]*(i64)(i - st[
12                top--;
```

```
13            }
14                st[top++] = i;
15            }
16        return ret;
17    }
```

ইনফাইনাইট চেসবোর্ডে নাইট ডিস্ট্যান্স [ইনডেক্স]

```
1    /*
2    NK is the size of grid you want to precalculate
3    NK/2,NK/2 will be considered origin
4    Calculates minimum knight distance from 0,0 to x,y
5    */
6
7    const int KN = 101;
8
9    i64 dk[KN][KN];
10   int dx[] = {-1, -1, 1, 1, -2, -2, 2, 2};
11   int dy[] = {-2, 2, -2, 2, -1, 1, -1, 1};
12
13   void precalc() {
14       int x, y, x1, y1, i;
15       queue< int > Q;
16       memset(dk, 0x3f, sizeof dk);
17       x = y = (KN >> 1);
18       dk[x][y] = 0;
19       Q.push(x); Q.push(y);
20       while(!Q.empty()) {
21           x = Q.front(); Q.pop();
22           y = Q.front(); Q.pop();
23           for(i = 0; i < 8; i++) {
24               x1 = x + dx[i], y1 = y + dy[i];
25               if(0 <= x1 && x1 < KN && 0 <= y1 && y1 < KN) {
26                   if(dk[x1][y1] > dk[x][y] + 1) {
27                       dk[x1][y1] = dk[x][y] + 1;
28                       Q.push(x1); Q.push(y1);
29                   }
30               }
31           }
32       }
33   }
34
35   i64 knight(i64 x, i64 y) {
36       i64 step, res = 0;
37       if(x < y) swap(x, y);
38       while((x<<1) > KN) {
39           step = x / 2 / 2; res += step;
40           x -= step * 2; y -= step;
41           if(y < 0) y = ((y % 2) + 2) % 2;
42           if(x < y) swap(x, y);
43       }
44       res += dk[x+(KN>>1)][y+(KN>>1)];
45       return res;
46   }
```

ট্রাই (প্রিফিক্স ট্রি) [ইনডেক্স]

```
1   /*
2   Basic trie, don't use it with large alphabet set or long len
3   All operation has complexity O(length).
4   MAX is number of different items.
5   */
6
7   struct trie {
8       trie *next[MAX+1];
9       trie() { for(int i=0; i<=MAX; i++) next[i] = NULL; }
10  };
11
12  void insert(trie *root, int *seq, int len) {
13      trie *curr = root;
14      for(int i = 0; i < len; i++) {
15          if(!curr->next[seq[i]]) curr->next[seq[i]] = new tri
16          curr = curr->next[seq[i]];
17      }
18      if(!curr->next[MAX]) curr->next[MAX] = new trie;
19  }
20
21  bool found(trie *root, int *seq, int len) {
22      trie *curr = root;
23      for(int i = 0; i < len; i++) {
24          if(!curr->next[seq[i]]) return false;
25          curr = curr->next[seq[i]];
26      }
27      if(!curr->next[MAX]) return false;
28      return true;
29  }
```

ট্রাই (স্ট্যাটিক অ্যারে) [ইনডেক্স]

```
1   /*
2   Trie implementation using array, faster and takes less memor
3   Each node can contain arbitrary data as needed for solving t
4   The ALPHABET, MAX and scale() may need tuning as necessary.
5   */
6
7   const int ALPHABET = 26;
8   const int MAX = 100000;
9
10  // for mapping items form 0 to ALPHABET-1
11  #define scale(x) (x-'a')
12
13
14  struct TrieTree {
15      int n, root;
16      int next[MAX][ALPHABET];
17      char data[MAX]; // there can be more data fields
18
19      void init() {
20          root = 0, n = 1; data[root] = 0;
21          memset(next[root], -1, sizeof(next[root]));
22      }
23
24      void insert(char *s) {
25          int curr = root, i, k;
26          for(i = 0; s[i]; i++) {
27              k = scale(s[i]);
```

```
28            if(next[curr][k] == -1) {
29                next[curr][k] = n;
30                data[n] = s[i]; // optional
31                memset(next[n], -1, sizeof(next[n]));
32                n++;
33            }
34            curr = next[curr][k];
35        }
36        data[curr] = 0; // sentinel, optional
37    }
38
39    bool find(char *s) {
40        int curr = root, i, k;
41        for(i = 0; s[i]; i++) {
42            k = scale(s[i]);
43            if(next[curr][k] == -1) return false;
44            curr = next[curr][k];
45        }
46        return (data[curr] == 0);
47    }
48
49 } trieTree;
```

সাফিক্স অ্যারে (সেলা) [ইনডেক্স]

```
1  /*
2  Generates suffix array in O(n (lg n)^2). Finds LCP in O(lg n
3  A[] is the target string with N length. base is the lowest c
4  S[i] holds the index of ith suffix when sorted lexicographic
5  Change base as necessary and MAXLOG = log2(MAXLEN).
6  */
7
8  int N, stp, P[MAXLOG][MAXLEN], S[MAXLEN];
9  char A[MAXLEN], base = 'a';
10 struct entry { int nr[2], p; } L[MAXLEN];
11
12 inline bool cmp(const entry &a, const entry &b) {
13     return a.nr[0] == b.nr[0] ? (a.nr[1] < b.nr[1] ? 1 : 0)
14 }
15
16 void generateSuffix() {
17     register int i, cnt;
18     for(i=0; i<N; i++) P[0][i] = A[i]-base;
19     for(stp = cnt = 1; cnt>>1 < N; stp++, cnt<<=1) {
20         for(i=0; i<N; i++) {
21             L[i].nr[0] = P[stp-1][i];
22             L[i].nr[1] = i+cnt<N ? P[stp-1][i+cnt] : -1;
23             L[i].p = i;
24         }
25         sort(L, L+N, cmp);
26         for(i=0; i<N; i++) {
27             if(i > 0 && L[i].nr[0] == L[i - 1].nr[0] && L[i]
28             else P[stp][L[i].p] = i;
29         }
30     }
31     for(i=0; i<N; i++) S[P[stp-1][i]] = i;
32 }
33
34 int lcp(int x, int y) {
```

```
35      int k, ret = 0;
36      if(x == y) return N - x;
37      for(k = stp - 1; k >= 0 && x < N && y < N; k --)
38          if(P[k][x] == P[k][y])
39              x += 1 << k, y += 1 << k, ret += 1 << k;
40      return ret;
41  }
```

সাফিক্স অ্যারে (n lg n) [ইনডেক্স]

```
1   /*
2   Suffix array implementation using bucket sorting + lcp.
3   Complexity O(n log n), str[] is the target string,
4   n is its length and suffix[i] contains i'th sorted suffix po
5   */
6
7   const int MAXN = 1 << 16;
8   const int MAXL = 16;
9
10  int n, stp, mv, suffix[MAXN], tmp[MAXN];
11  int sum[MAXN], cnt[MAXN], rank[MAXL][MAXN];
12  char str[MAXN];
13
14  inline bool equal(const int &u, const int &v){
15      if(!stp) return str[u] == str[v];
16      if(rank[stp-1][u] != rank[stp-1][v]) return false;
17      int a = u + mv < n ? rank[stp-1][u+mv] : -1;
18      int b = v + mv < n ? rank[stp-1][v+mv] : -1;
19      return a == b;
20  }
21
22  void update(){
23      int i, rnk;
24      for(i = 0; i < n; i++) sum[i] = 0;
25      for(i = rnk = 0; i < n; i++) {
26          suffix[i] = tmp[i];
27          if(i && !equal(suffix[i], suffix[i-1])) {
28              rank[stp][suffix[i]] = ++rnk;
29              sum[rnk+1] = sum[rnk];
30          }
31          else rank[stp][suffix[i]] = rnk;
32          sum[rnk+1]++;
33      }
34  }
35
36  void Sort() {
37      int i;
38      for(i = 0; i < n; i++) cnt[i] = 0;
39      memset(tmp, -1, sizeof tmp);
40      for(i = 0; i < mv; i++){
41          int idx = rank[stp - 1][n - i - 1];
42          int x = sum[idx];
43          tmp[x + cnt[idx]] = n - i - 1;
44          cnt[idx]++;
45      }
46      for(i = 0; i < n; i++){
47          int idx = suffix[i] - mv;
48          if(idx < 0)continue;
49          idx = rank[stp-1][idx];
```

```
50            int x = sum[idx];
51            tmp[x + cnt[idx]] = suffix[i] - mv;
52            cnt[idx]++;
53        }
54        update();
55        return;
56    }
57
58    inline bool cmp(const int &a, const int &b){
59        if(str[a]!=str[b]) return str[a]<str[b];
60        return false;
61    }
62
63    void SortSuffix() {
64        int i;
65        for(i = 0; i < n; i++) tmp[i] = i;
66        sort(tmp, tmp + n, cmp);
67        stp = 0;
68        update();
69        ++stp;
70        for(mv = 1; mv < n; mv <<= 1) {
71            Sort();
72            stp++;
73        }
74        stp--;
75        for(i = 0; i <= stp; i++) rank[i][n] = -1;
76    }
77
78    inline int lcp(int u, int v) {
79        if(u == v) return n - u;
80        int ret, i;
81        for(ret = 0, i = stp; i >= 0; i--) {
82            if(rank[i][u] == rank[i][v]) {
83                ret += 1<<i;
84                u += 1<<i;
85                v += 1<<i;
86            }
87        }
88        return ret;
89    }
```

মিনিমাম এক্সপ্রেশন (ঝো-জুয়ান) [ইনডেক্স]

```
1    /*
2    Finds alphabetically first representation of a cyclic string
3    */
4
5    inline int minimumExpression(char *s) {
6        int i, j, k, n, len, p, q;
7        len = n = strlen(s), n <<= 1, i = 0, j = 1, k = 0;
8        while(i + k < n && j + k < n) {
9            p = i+k >= len ? s[i+k-len] : s[i+k];
10           q = j+k >= len ? s[j+k-len] : s[j+k];
11           if(p == q) k++;
12           else if(p > q) { i = i+k+1; if(i <= j) i = j+1; k =
13           else if(p < q) { j = j+k+1; if(j <= i) j = i+1; k =
14       }
15       return i < j ? i : j;
16    }
```

কে.এম.পি [ইনডেক্স]

```
1    /*
2    finds all occurance of pattern in buffer
3    buffer can be online
4    */
5
6    char buffer[MAX], pattern[MAX];
7    int overlap[MAX];
8
9    void computeOverlap() {
10       overlap[0] = -1;
11       for(int i=0; pattern[i]; i++) {
12           overlap[i+1] = overlap[i] + 1;
13           while(overlap[i+1] > 0 && pattern[i] != pattern[over
14               overlap[i+1] = overlap[overlap[i+1]-1] + 1;
15           }
16       }
17   }
18
19   vector< int > kmpMatcher(int m) {
20       vector< int > V;
21       for(int i = 0, j = 0; buffer[i]; i++) {
22           while(true) {
23               if(buffer[i] == pattern[j]) {
24                   j++;
25                   if(j == m) {
26                       V.push_back(i-m+1);
27                       j = overlap[j];
28                   }
29                   break;
30               }
31               else if(j == 0) break;
32               else j = overlap[j];
33           }
34       }
35       return V;
36   }
```

ম্যানাক্যার আলগোরিদম [ইনডেক্স]

```
1    /*
2    Manacher algorithm implementation.
3    Application, largest palindromic substring, largest palindro
4    */
5
6    int lengths[MAX<<1];
7
8    int manacher(char *buff, int len) {
9        int i, k, pallen, found, d, j, s, e;
10       k = pallen = 0;
11       for(i = 0; i < len; ) {
12           if(i > pallen && buff[i-pallen-1] == buff[i]) {
13               pallen += 2, i++;
14               continue;
15           }
16           lengths[k++] = pallen;
```

```
17          s = k - 2, e = s - pallen, found = 0;
18          for(j = s; j > e; j--) {
19              d = j - e - 1;
20              if(lengths[j] == d) {
21                  pallen = d;
22                  found = 1;
23                  break;
24              }
25              lengths[k++] = (d < lengths[j]? d : lengths[j]);
26          }
27          if(!found) { pallen = 1; i++; }
28      }
29      lengths[k++] = pallen;
30      return lengths[k-1];
31  }
```

জোসেফাস রিকারেন্স [ইনডেক্স]

```
1   /*
2   The first one is for K = 2 and the second one is general.
3   Note: first function returns 1 based index while second one
4   */
5
6   int f(int n) {
7       if(n == 1) return 1;
8       return (f((n-(n&1))>>1)<<1) + ((n&1)?1:-1);
9   }
10
11  int f(int n, int k) {
12      if(n == 1) return 0;
13      return (f(n-1, k) + k)%n;
14  }
```

শ্যাঙ্ক’স বেবি স্টেপ জায়ান্ট স্টেপ [ইনডেক্স]

```
1   /*
2   Shanks baby step giant step - discrete logarithm algorithm
3   for the equation: b = a^x % p where a, b, p known, finds x
4   works only when p is an odd prime
5   */
6
7   int shank(int a, int b, int p) {
8       int i, j, m;
9       long long c, aj, ami;
10      map< long long, int > M;
11      map< long long, int > :: iterator it;
12      m = (int)ceil(sqrt((double)(p)));
13      M.insert(make_pair(1, 0));
14      for(j = 1, aj = 1; j < m; j++) {
15          aj = (aj * a) % p;
16          M.insert(make_pair(aj, j));
17      }
18      ami = modexp(modinv(a, p), m, p);
19      for(c = b, i = 0; i < m; i++) {
20          it = M.find(c);
21          if(it != M.end()) return i * m + it->second;
22          c = (c * ami) % p;
```

```
23        }
24        return 0;
25    }
```

সেগমেন্টেড সিভ [ইনডেক্স]

```
1  /*
2  Generates primes within interval [a, b] when b - a <= 100000
3  and 1 <= a <= b <= 2147483647
4  */
5  int base[MAX>>6], segment[RNG>>6], primes[LEN], prlen;
6
7  #define chkC(x,n) (x[n>>6]&(1<<((n>>1)&31)))
8  #define setC(x,n) (x[n>>6]|=(1<<((n>>1)&31)))
9
10 void sieve() {
11     int i, j, k;
12     for(i=3; i<LMT; i+=2) if(!chkC(base, i)) for(j=i*i, k=i<
13     for(i=3, prlen=0; i<MAX; i+=2) if(!chkC(base, i)) primes
14 }
15
16 int segmented_sieve(int a, int b) {
17     int rt, i, k, cnt = (a<=2 && 2<=b)? 1 : 0;
18     if(b<2) return 0;
19     if(a<3) a = 3;
20     if(a%2==0) a++;
21     memset(segment, 0, sizeof segment);
22     for(i=0, rt=(int)sqrt((double)b); i < prlen && primes[i]
23         unsigned j = primes[i] * ( (a+primes[i]-1) / primes[
24         if(j%2==0) j += primes[i];
25         for(k=primes[i]<<1; j<=b; j+=k) if(j!=primes[i]) set
26     }
27     for(i=0; i<=b-a; i+=2) if(!chkC(segment, i)) cnt++;
28     return cnt;
29 }
```

এক্সটেন্ডেড ইউক্লিড ও মডুলার ইনভার্স [ইনডেক্স]

```
1  /*
2  Implementation of extended euclid algorithm.
3  Modular inverse requires gcd(a, n) = 1.
4  */
5
6  class Euclid {
7  public:
8      i64 x, y, d;
9      Euclid() {}
10     Euclid(i64 _x, i64 _y, i64 _d) : x(_x), y(_y), d(_d) {}
11 };
12
13 Euclid egcd(i64 a, i64 b) {
14     if(!b) return Euclid(1, 0, a);
15     Euclid r = egcd(b, a % b);
16     return Euclid(r.y, r.x - a / b * r.y, r.d);
17 }
18
19 i64 modinv(i64 a, i64 n) {
```

```
20        Euclid t = egcd(a, n);
21        if(t.d > 1) return 0;
22        i64 r = t.x % n;
23        return (r < 0 ? r + n : r);
24    }
```

কনভেক্স হাল (গ্রাহাম'স স্ক্যান) [ইনডেক্স]

```
1   /*
2   ConvexHull : Graham's Scan O(n lg n), integer implementation
3   P[]: holds all the points, C[]: holds points on the hull
4   np: number of points in P[], nc: number of points in C[]
5   to handle duplicate, call makeUnique() before calling convex
6   call convexHull() if you have np >= 3
7   to remove co-linear points on hull, call compress() after co
8   */
9
10  point P[MAX], C[MAX], P0;
11
12  inline int triArea2(const point &a, const point &b, const po
13      return (a.x*(b.y-c.y) + b.x*(c.y-a.y) + c.x*(a.y-b.y));
14  }
15
16  inline int sqDist(const point &a, const point &b) {
17      return ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
18  }
19
20  inline bool comp(const point &a, const point &b) {
21      int d = triArea2(P0, a, b);
22      if(d < 0) return false;
23      if(!d && sqDist(P0, a) > sqDist(P0, b)) return false;
24      return true;
25  }
26
27  inline bool normal(const point &a, const point &b) {
28      return ((a.x==b.x) ? a.y < b.y : a.x < b.x);
29  }
30
31  inline bool issame(const point &a, const point &b) {
32      return (a.x == b.x && a.y == b.y);
33  }
34
35  inline void makeUnique(int &np) {
36      sort(&P[0], &P[np], normal);
37      np = unique(&P[0], &P[np], issame) - P;
38  }
39
40  void convexHull(int &np, int &nc) {
41      int i, j, pos = 0;
42      for(i = 1; i < np; i++)
43          if(P[i].y<P[pos].y || (P[i].y==P[pos].y && P[i].x<P[
44              pos = i;
45      swap(P[0], P[pos]);
46      P0 = P[0];
47      sort(&P[1], &P[np], comp);
48      for(i = 0; i < 3; i++) C[i] = P[i];
49      for(i = j = 3; i < np; i++) {
50          while(triArea2(C[j-2], C[j-1], P[i]) < 0) j--;
51          C[j++] = P[i];
```

```
52          }
53      nc = j;
54   }
55
56   void compress(int &nc) {
57       int i, j, d;
58       C[nc] = C[0];
59       for(i=j=1; i < nc; i++) {
60           d = triArea2(C[j-1], C[i], C[i+1]);
61           if(d || (!d && issame(C[j-1], C[i+1]))) C[j++] = C[i
62       }
63       nc = j;
64   }
```

পয়েন্ট ইন কনভেক্স পলিগন [ইনডেক্স]

```
1    /*
2    C[] array of points of convex polygon in ccw order,
3    nc number of points in C, p target points.
4    returns true if p is inside C (including edge) or false othe
5    complexity O(lg n)
6    */
7
8    inline bool inConvexPoly(point *C, int nc, const point &p) {
9        int st = 1, en = nc - 1, mid;
10       while(en - st > 1) {
11           mid = (st + en)>>1;
12           if(triArea2(C[0], C[mid], p) < 0) en = mid;
13           else st = mid;
14       }
15       if(triArea2(C[0], C[st], p) < 0) return false;
16       if(triArea2(C[st], C[en], p) < 0) return false;
17       if(triArea2(C[en], C[0], p) < 0) return false;
18       return true;
19   }
```

পলিগন এরিয়া (2D) [ইনডেক্স]

```
1    /*
2    P[] holds the points, must be either in cw or ccw
3    function returns double of the area.
4    */
5
6    inline int dArea(int np) {
7        int area = 0;
8        for(int i = 0; i < np; i++) {
9            area += p[i].x*p[i+1].y - p[i].y*p[i+1].x;
10       }
11       return abs(area);
12   }
```

ক্লোজেস্ট পেয়ার [ইনডেক্স]

```
1    /*
2    closestPair(Point *X, Point *Y, int n);
3    X contains the points sorted by x co-ordinate,
```

```
 4    Y contains the points sorted by y co-ordinate,
 5    One additional item in Point structure is needed, the origin
 6    */
 7
 8    typedef long long i64;
 9    typedef struct { int x, y, i; } Point;
10
11    int flag[MAX];
12
13    inline i64 sq(const i64 &x) {
14        return x*x;
15    }
16
17    inline i64 sqdist(const Point &a, const Point &b) {
18        return sq(a.x-b.x) + sq(a.y-b.y);
19    }
20
21    inline i64 closestPair(Point *X, Point *Y, int n) {
22        if(n == 1) return INF;
23        if(n == 2) return sqdist(X[0], X[1]);
24
25        int i, j, k, n1, n2, ns, m = n >> 1;
26        Point Xm = X[m-1], *XL, *XR, *YL, *YR, *YS;
27        i64 lt, rt, dd, tmp;
28
29        XL = new Point[m], YL = new Point[m];
30        XR = new Point[m+1], YR = new Point[m+1];
31        YS = new Point[n];
32
33        for(i = 0; i < m; i++) XL[i] = X[i], flag[X[i].i] = 0;
34        for(; i < n; i++) XR[i - m] = X[i], flag[X[i].i] = 1;
35        for(i = n2 = n1 = 0; i < n; i++) {
36            if(!flag[Y[i].i]) YL[n1++] = Y[i];
37            else YR[n2++] = Y[i];
38        }
39
40        lt = closestPair(XL, YL, n1);
41        rt = closestPair(XR, YR, n2);
42        dd = min(lt, rt);
43
44        for(i = ns = 0; i < n; i++)
45            if(sq(Y[i].x - Xm.x) < dd)
46                YS[ns++] = Y[i];
47        for(j = 0; j < ns; j++)
48            for(k = j + 1; k < ns && sq(YS[k].y - YS[j].y) < dd;
49                dd = min(dd, sqdist(YS[j], YS[k])));
50
51        delete[] XL; delete[] XR;
52        delete[] YL; delete[] YR;
53        delete[] YS;
54
55        return dd;
56    }
```

সার্কেল ইন্টারসেকশন এরিয়া [ইনডেক্স]

```
 1    /*
 2    This code assumes the circle center and radius to be integer
 3    Change this when necessary.
```

```
4    */
5
6    inline double commonArea(const Circle &a, const Circle &b) {
7        int dsq = sqDist(a.c, b.c);
8        double d = sqrt((double)dsq);
9        if(sq(a.r + b.r) <= dsq) return 0;
10       if(a.r >= b.r && sq(a.r-b.r) >= dsq) return pi * b.r * b
11       if(a.r <= b.r && sq(b.r-a.r) >= dsq) return pi * a.r * a
12       double angleA = 2.0 * acos((a.r * a.r + dsq - b.r * b.r)
13       double angleB = 2.0 * acos((b.r * b.r + dsq - a.r * a.r)
14       return 0.5 * (a.r * a.r * (angleA - sin(angleA)) + b.r *
15   }
```

সেগমেন্ট ইন্টারসেকশন (2D) [ইনডেক্স]

```
1    /*
2    Segment intersection in 2D integer space.
3    P1, p2 makes first segment, p3, p4 makes the second segment
4    */
5
6    inline bool intersect(const Point &p1, const Point &p2, cons
7        i64 d1, d2, d3, d4;
8        d1 = direction(p3, p4, p1);
9        d2 = direction(p3, p4, p2);
10       d3 = direction(p1, p2, p3);
11       d4 = direction(p1, p2, p4);
12       if(((d1 < 0 && d2 > 0) || (d1 > 0 && d2 < 0)) && ((d3 <
13       if(!d3 && onsegment(p1, p2, p3)) return true;
14       if(!d4 && onsegment(p1, p2, p4)) return true;
15       if(!d1 && onsegment(p3, p4, p1)) return true;
16       if(!d2 && onsegment(p3, p4, p2)) return true;
17       return false;
18   }
```

টেট্রাহেড্রন ফরমুলা [ইনডেক্স]

```
1    /*
2    Some tetrahedron formulas
3    */
4
5    inline double volume(double u, double v, double w, double U,
6        double u1,v1,w1;
7        u1 = v * v + w * w - U * U;
8        v1 = w * w + u * u - V * V;
9        w1 = u * u + v * v - W * W;
10       return sqrt(4.0*u*u*v*v*w*w - u*u*u1*u1 - v*v*v1*v1 - w*
11   }
12
13   inline double surface(double a, double b, double c) {
14       return sqrt((a + b + c) * (-a + b + c) * (a - b + c) * (
15   }
16
17   inline double insphere(double WX, double WY, double WZ, doub
18       double sur, rad;
19       sur = surface(WX, WY, XY) + surface(WX, XZ, WZ) + surfac
20       rad = volume(WX, WY, WZ, YZ, XZ, XY) * 3.0 / sur;
21       return rad;
```

```
22   }
```

স্ট্রংলি কানেকটেড কম্পোনেন্ট (টারজান) [ইনডেক্স]

```
1    /*
2    SCC (Tarjan) in O(|v| + |e|)
3    Input:
4    G[] is a input directed graph with n nodes in range [1,n]
5    Output:
6    Component[i] holds the component id to which node i belongs
7    components: total number of components in the graph
8    */
9
10   int Stack[MAX], top;
11   int Index[MAX], Lowlink[MAX], Onstack[MAX];
12   int Component[MAX];
13   int idx, components;
14   vector< int > G[MAX];
15
16   void tarjan(int u) {
17       int v, i;
18       Index[u] = Lowlink[u] = idx++;
19       Stack[top++] = u;
20       Onstack[u] = 1;
21       for(i = 0; i < SZ(G[u]); i++) {
22           v = G[u][i];
23           if(Index[v]==-1) {
24               tarjan(v);
25               Lowlink[u] = min(Lowlink[u], Lowlink[v]);
26           }
27           else if(Onstack[v]) Lowlink[u] = min(Lowlink[u], Ind
28       }
29       if(Lowlink[u] == Index[u]) {
30           components++;
31           do {
32               v = Stack[--top];
33               Onstack[v] = 0;
34               Component[v] = components;
35           } while(u != v);
36       }
37   }
38
39   void findSCC(int n) {
40       components = top = idx = 0;
41       SET(Index); CLR(Onstack); MEM(Lowlink, 0x3f);
42       for(int i = 1; i <= n; i++) if(Index[i]==-1) tarjan(i);
43   }
```

আর্টিকুলেশন পয়েন্ট [ইনডেক্স]

```
1    /*
2    G[][]: undirected graph
3    cut[v] is true if node v is an articulation point / cut-vert
4    */
5
6    vector< int > G[MAX];
7    int low[MAX], vis[MAX], used[MAX], cut[MAX], dfstime;
```

```
8
9    void dfs(int u, int par = -1) {
10       int i, v, child = 0;
11       used[u] = 1;
12       vis[u] = low[u] = ++dfstime;
13       for(i = 0; i < G[u].size(); i++) {
14          v = G[u][i];
15          if(v == par) continue;
16          if(used[v]) low[u] = min(low[u], vis[v]);
17          else {
18             child++;
19             dfs(v, u);
20             low[u] = min(low[u], low[v]);
21             if(low[v] >= vis[u]) cut[u] = 1;
22          }
23       }
24       if(par == -1) cut[u] = (child > 1);
25    }
```

ব্রিজ [ইনডেক্স]

```
1    /*
2    G[][]: undirected graph
3    finds all the bridges in a connected graph and
4    adds those edges to the Bridges[] vector
5    */
6
7    vector< int > G[MAX];
8    vector< pair< int, int > > Bridges;
9    int low[MAX], vis[MAX], used[MAX], dfstime;
10
11   void dfs(int u, int par) {
12       int i, v;
13       used[u] = 1;
14       vis[u] = low[u] = ++dfstime;
15       for(i = 0; i < G[u].size(); i++) {
16          v = G[u][i];
17          if(v == par) continue;
18          if(used[v]) low[u] = min(low[u], vis[v]);
19          else {
20             dfs(v, u);
21             low[u] = min(low[u], low[v]);
22             if(low[v] > vis[u]) Bridges.push_back(make_pair(
23          }
24       }
25   }
```

বাই কানেকটেড কম্পোনেন্ট [ইনডেক্স]

```
1    /*
2    G[][]: undirected graph
3    Separates bi-connected component by edges.
4    */
5
6    vector< int > G[MAX];
7    stack< pii > S;
8    int dfstime;
```

```
 9    int low[MAX], vis[MAX], used[MAX];
10
11    void dfs(int u, int par) {
12        int v, i, sz = G[u].size();
13        pii e, curr;
14        used[u] = 1;
15        vis[u] = low[u] = ++dfstime;
16        for(i = 0; i < sz; i++) {
17            v = G[u][i];
18            if(v == par) continue;
19            if(!used[v]) {
20                S.push(pii(u, v));
21                dfs(v, u);
22                if(low[v] >= vis[u]) {
23                    // new component
24                    curr = pii(u, v);
25                    do {
26                        e = S.top(); S.pop();
27                        // e is an edge in current bcc
28                    } while(e != curr);
29                }
30                low[u] = min(low[u], low[v]);
31            }
32            else if(vis[v] < vis[u]) {
33                S.push(pii(u, v));
34                low[u] = min(low[u], vis[v]);
35            }
36        }
37    }
```

স্টেবল ম্যারিজ [ইনডেক্স]

```
 1    /*
 2    INPUT:
 3    m: number of man, n: number of woman (must be at least as la
 4    L[i][]: the list of women in order of decreasing preference
 5    R[j][i]: the attractiveness of i to j.
 6    OUTPUTS:
 7    L2R[]: the mate of man i (always between 0 and n-1)
 8    R2L[]: the mate of woman j (or -1 if single)
 9    man priority
10    */
11
12    int m, n, L[MAXM][MAXW], R[MAXW][MAXM], L2R[MAXM], R2L[MAXW]
13
14    void stableMarriage() {
15        int i, man, wom, hubby;
16        SET(R2L); CLR(p);
17        for(i = 0; i < m; i++ ) {
18            man = i;
19            while(man >= 0) {
20                while(true) {
21                    wom = L[man][p[man]++];
22                    if(R2L[wom] < 0 || R[wom][man] > R[wom][R2L[
23                }
24                hubby = R2L[wom];
25                R2L[L2R[man] = wom] = man;
26                man = hubby;
27            }
28        }
```

```
29        }
```

ম্যক্স ফ্লো (ডিনিক) [ইনডেক্স]

```
1    /*
2    max flow (dinitz algorithm)
3    works on undirected graph
4    can have loops, multiple edges, cycles
5    */
6
7    int src, snk, nNode, nEdge;
8    int Q[MAXN], fin[MAXN], pro[MAXN], dist[MAXN];
9    int flow[MAXE], cap[MAXE], next[MAXE], to[MAXE];
10
11   inline void init(int _src, int _snk, int _n) {
12       src = _src, snk = _snk, nNode = _n, nEdge = 0;
13       SET(fin);
14   }
15
16   inline void add(int u, int v, int _cap) {
17       to[nEdge] = v, cap[nEdge] = _cap, flow[nEdge] = 0;
18       next[nEdge] = fin[u], fin[u] = nEdge++;
19       to[nEdge] = u, cap[nEdge] = _cap, flow[nEdge] = 0;
20       next[nEdge] = fin[v], fin[v] = nEdge++;
21   }
22
23   bool bfs() {
24       int st, en, i, u, v;
25       SET(dist);
26       dist[src] = st = en = 0;
27       Q[en++] = src;
28       while(st < en) {
29           u = Q[st++];
30           for(i=fin[u]; i>=0; i=next[i]) {
31               v = to[i];
32               if(flow[i] < cap[i] && dist[v]==-1) {
33                   dist[v] = dist[u]+1;
34                   Q[en++] = v;
35               }
36           }
37       }
38       return dist[snk]!=-1;
39   }
40
41   int dfs(int u, int fl) {
42       if(u==snk) return fl;
43       for(int &e=pro[u], v, df; e>=0; e=next[e]) {
44           v = to[e];
45           if(flow[e] < cap[e] && dist[v]==dist[u]+1) {
46               df = dfs(v, min(cap[e]-flow[e], fl));
47               if(df>0) {
48                   flow[e] += df;
49                   flow[e^1] -= df;
50                   return df;
51               }
52           }
53       }
54       return 0;
55   }
```

```
56
57   i64 dinitz() {
58       i64 ret = 0;
59       int df;
60       while(bfs()) {
61           for(int i=1; i<=nNode; i++) pro[i] = fin[i];
62           while(true) {
63               df = dfs(src, INF);
64               if(df) ret += (i64)df;
65               else break;
66           }
67       }
68       return ret;
69   }
```

মিন কস্ট ম্যক্সস ফ্লো (বেলম্যান ফোর্ড) [ইনডেক্স]

```
1    /*
2    min cost flow (bellman ford)
3    works only on directed graphs
4    handles multiple edges, cycles, loops
5    */
6
7    int src, snk, nNode, nEdge;
8    int fin[MAXN], pre[MAXN], dist[MAXN];
9    int cap[MAXE], cost[MAXE], next[MAXE], to[MAXE], from[MAXE];
10
11   inline void init(int _src, int _snk, int nodes) {
12       SET(fin);
13       nNode = nodes, nEdge = 0;
14       src = _src, snk = _snk;
15   }
16
17   inline void addEdge(int u, int v, int _cap, int _cost) {
18       from[nEdge] = u, to[nEdge] = v, cap[nEdge] = _cap, cost[
19       next[nEdge] = fin[u], fin[u] = nEdge++;
20       from[nEdge] = v, to[nEdge] = u, cap[nEdge] = 0, cost[nEd
21       next[nEdge] = fin[v], fin[v] = nEdge++;
22   }
23
24   bool bellman() {
25       int iter, u, v, i;
26       bool flag = true;
27       MEM(dist, 0x7f);
28       SET(pre);
29       dist[src] = 0;
30       for(iter = 1; iter < nNode && flag; iter++) {
31           flag = false;
32           for(u = 0; u < nNode; u++) {
33               for(i = fin[u]; i >= 0; i = next[i]) {
34                   v = to[i];
35                   if(cap[i] && dist[v] > dist[u] + cost[i]) {
36                       dist[v] = dist[u] + cost[i];
37                       pre[v] = i;
38                       flag = true;
39                   }
40               }
41           }
42       }
43       return (dist[snk] < INF);
```

```
44    }
45
46    int mcmf(int &fcost) {
47        int netflow, i, bot, u;
48        netflow = fcost = 0;
49        while(bellman()) {
50            bot = INF;
51            for(u = pre[snk]; u >= 0; u = pre[from[u]]) bot = mi
52            for(u = pre[snk]; u >= 0; u = pre[from[u]]) {
53                cap[u] -= bot;
54                cap[u^1] += bot;
55                fcost += bot * cost[u];
56            }
57            netflow += bot;
58        }
59        return netflow;
60    }
```

ম্যাক্সিমাম বাইপারটাইট ম্যাচিং (ডি.এফ.এস.) [ইনডেক্স]

```
1     /*
2     G[] is the left-side graph, must be bipartite
3     match(n): n is the number of nodes in left-side set
4     and returns the maximum possible matching.
5     Left[] anf Right[] ar assigned with corresponding matches
6     */
7
8     vector < int > G[MAX];
9     bool visited[MAX];
10    int Left[MAX], Right[MAX];
11
12    bool dfs(int u) {
13        if(visited[u]) return false;
14        visited[u] = true;
15        int len = G[u].size(), i, v;
16        for(i=0; i<len; i++) {
17            v = G[u][i];
18            if(Right[v]==-1) {
19                Right[v] = u, Left[u] = v;
20                return true;
21            }
22        }
23        for(i=0; i<len; i++) {
24            v = G[u][i];
25            if(dfs(Right[v])) {
26                Right[v] = u, Left[u] = v;
27                return true;
28            }
29        }
30        return false;
31    }
32
33    int match(int n) {
34        int i, ret = 0;
35        bool done;
36        SET(Left); SET(Right);
37        do {
38            done = true; CLR(visited);
39            for(i=0; i<n; i++) {
```

```
40              if(Left[i]==-1 && dfs(i)) {
41                  done = false;
42              }
43          }
44      } while(!done);
45      for(i=0; i<n; i++) ret += (Left[i]!=-1);
46      return ret;
47  }
```

ম্যাক্সিমাম বাইপারটাইট ম্যাচিং (হপক্রফট কার্প) [ইনডেক্স]

```
1   /*
2   n: number of nodes on left side, nodes are numbered 1 to n
3   m: number of nodes on right side, nodes are numbered n+1 to
4   G = NIL[0] ? G1[G[1---n]] ? G2[G[n+1---n+m]]
5   */
6
7   bool bfs() {
8       int i, u, v, len;
9       queue< int > Q;
10      for(i=1; i<=n; i++) {
11          if(match[i]==NIL) {
12              dist[i] = 0;
13              Q.push(i);
14          }
15          else dist[i] = INF;
16      }
17      dist[NIL] = INF;
18      while(!Q.empty()) {
19          u = Q.front(); Q.pop();
20          if(u!=NIL) {
21              len = G[u].size();
22              for(i=0; i<len; i++) {
23                  v = G[u][i];
24                  if(dist[match[v]]==INF) {
25                      dist[match[v]] = dist[u] + 1;
26                      Q.push(match[v]);
27                  }
28              }
29          }
30      }
31      return (dist[NIL]!=INF);
32  }
33
34  bool dfs(int u) {
35      int i, v, len;
36      if(u!=NIL) {
37          len = G[u].size();
38          for(i=0; i<len; i++) {
39              v = G[u][i];
40              if(dist[match[v]]==dist[u]+1) {
41                  if(dfs(match[v])) {
42                      match[v] = u;
43                      match[u] = v;
44                      return true;
45                  }
46              }
47          }
48          dist[u] = INF;
49          return false;
```

```
50          }
51          return true;
52      }
53
54      int hopcroft_karp() {
55          int matching = 0, i;
56          CLR(match);
57          while(bfs())
58              for(i=1; i<=n; i++)
59                  if(match[i]==NIL && dfs(i))
60                      matching++;
61          return matching;
62      }
```

হেভী লাইট ডিকম্পজিশন [ইনডেক্স]

```
1   /*
2   Rough code for heavy light decomposition. Input graph must b
3   Also includes the LCA method.
4   What to do with the chain is problem dependent.
5   */
6
7   vector< int > G[MAX];
8   int cost[MAX], lvl[MAX], parent[MAX];
9   int head[MAX], cnext[MAX], chainid[MAX], chainpos[MAX];
10  int nchain, temp[MAX];
11
12  int dfs(int u, int p, int d) {
13      int i, v, sz = G[u].size(), tmp, mx, id, tot, hd, k;
14      lvl[u] = d, mx = 0, id = u, tot = 1;
15      for(i = 0; i < sz; i++) {
16          v = G[u][i];
17          if(v != p) {
18              parent[v] = u;
19              tmp = dfs(v, u, d + 1);
20              tot += tmp;
21              if(tmp > mx) {
22                  mx = tmp;
23                  id = v;
24              }
25          }
26      }
27      if(tot == 1) cnext[u] = -1;
28      else cnext[u] = id;
29      for(i = 0; i < sz; i++) {
30          v = G[u][i];
31          if(v != p && v != id) {
32              for(hd = v, k = 0; v != -1; v = cnext[v], k++) {
33                  head[v] = hd;
34                  temp[k] = cost[v];
35                  chainpos[v] = k;
36                  chainid[v] = nchain;
37              }
38              // buff is the current chain of size k
39              nchain++;
40          }
41      }
42      return tot;
43  }
```

```
44
45    void hld(int v) {
46        int hd, k;
47        nchain = 0; lvl[0] = -1;
48        dfs(v, 0, 0);
49        for(hd = v, k = 0; v != -1; v = cnext[v], k++) {
50            head[v] = hd;
51            temp[k] = cost[v];
52            chainpos[v] = k;
53            chainid[v] = nchain;
54        }
55        // buff is the current chain of size k
56        nchain++;
57    }
58
59    int lca(int a, int b) {
60        while(chainid[a] != chainid[b]) {
61            if(lvl[head[a]] < lvl[head[b]]) b = parent[head[b]];
62            else a = parent[head[a]];
63        }
64        return (lvl[a] < lvl[b]) ? a : b;
65    }
```

টারজান'স অফলাইন এল.সি.এ [ইনডেক্স]

```
1    /*
2    Tarjan's offline LCA algorithm. For each pair of node's in P
3    it finds the LCA of the nodes in the rooted tree G (no edge
4    The array ans holds the result for queries in orders defined
5    */
6
7    void lca(int u) {
8        int v, i, sz;
9        make_set(u);
10       ancestor[find_set(u)] = u;
11       sz = G[u].size();
12       for(i = 0; i < sz; i++) {
13           v = G[u][i];
14           lca(v);
15           union_set(u, v);
16           ancestor[find_set(u)] = u;
17       }
18       color[u] = 1;
19       sz = P[u].size();
20       for(i = 0; i < sz; i++) {
21           v = P[u][i].first;
22           if(color[v]) ans[P[u][i].second] = ancestor[find_set
23       }
24    }
```

বাইনারি সার্চ [ইনডেক্স]

```
1    /*
2    st and en defines the range [st, en) in a[]
3    in every range, start is inclusive and end is exclusive
4    */
5
```

```
 6    // returns index of first element not less than val
 7    int lowerbound(int *a, int st, int en, int val) {
 8        int idx, cnt, stp;
 9        cnt = en - st;
10        while(cnt > 0) {
11            stp = cnt >> 1; idx = st + stp;
12            if(a[idx] < val) st = ++idx, cnt -= stp+1;
13            else cnt = stp;
14        }
15        return st;
16    }
17
18    // returns index of first element greater than val
19    int upperbound(int *a, int st, int en, int val) {
20        int idx, cnt, stp;
21        cnt = en - st;
22        while(cnt > 0) {
23            stp = cnt >> 1; idx = st + stp;
24            if(a[idx] <= val) st = ++idx, cnt -= stp+1;
25            else cnt = stp;
26        }
27        return st;
28    }
29
30    // returns true if items is found, otherwise false
31    bool binarysearch(int *a, int st, int en, int val) {
32        int lb = lowerbound(a, st, en, val);
33        return (lb != en && a[lb] == val);
34    }
35
36    // returns the interval where each element is equal to val
37    pair< int, int > equalrange(int *a, int st, int en, int val)
38        int ub = upperbound(a, st, en, val);
39        int lb = lowerbound(a, st, en, val);
40        return pair< int, int >(lb, ub);
41    }
```

ডিসজয়েন্ট সেট [ইনডেক্স]

```
 1    /*
 2    disjoint set data-structure
 3    implements union by rank and path compression
 4    */
 5
 6    struct DisjointSet {
 7        int *root, *rank, n;
 8        DisjointSet(int sz) {
 9            root = new int[sz+1];
10            rank = new int[sz+1];
11            n = sz;
12        }
13        ~DisjointSet() {
14            delete[] root;
15            delete[] rank;
16        }
17        void init() {
18            for(int i = 1; i <= n; i++) {
19                root[i] = i;
20                rank[i] = 0;
```

```
21              }
22          }
23      int find(int u) {
24          if(u != root[u]) root[u] = find(root[u]);
25          return root[u];
26      }
27      void merge(int u, int v) {
28          int pu = find(u);
29          int pv = find(v);
30          if(rank[pu] > rank[pv]) root[pv] = pu;
31          else root[pu] = pv;
32          if(rank[pu]==rank[pv]) rank[pv]++;
33      }
34  };
```

2D বি.আই.টি [ইনডেক্স]

```
1   /*
2   update and query function for 2D bit.
3   MAX is the maximum possible value.
4   bit[][] holds the 2D binary indexed tree
5   */
6
7   void update(int x, int y, int v) {
8       int y1;
9       while(x <= MAX) {
10          y1 = y;
11          while(y1 <= MAX) {
12              bit[x][y1] += v;
13              y1 += (y1 & -y1);
14          }
15          x += (x & -x);
16      }
17  }
18
19  int readsum(int x, int y) {
20      int v = 0, y1;
21      while(x > 0) {
22          y1 = y;
23          while(y1 > 0) {
24              v += bit[x][y1];
25              y1 -= (y1 & -y1);
26          }
27          x -= (x & -x);
28      }
29      return v;
30  }
```

ডাবল হ্যাশিং [ইনডেক্স]

```
1   /*
2   M > N and should be close, better both be primes.
3   M should be as much large as possible, not exceeding array s
4   HKEY is the Hash function, change it if necessary.
5   */
6
7   #define NIL -1
8   #define M 1021
9   #define N 1019
```

```
10    #define HKEY(x,i) ((x)%M+(i)*(1+(x)%N))%M
11
12    int a[M+1];
13
14    inline int hash(int key) {
15        int i = 0, j;
16        do {
17            j = HKEY(key, i);
18            if(a[j]==NIL) { a[j] = key; return j; }
19            i++;
20        } while(i < M);
21        return -1;
22    }
23
24    inline int find(int key) {
25        int i = 0, j;
26        do {
27            j = HKEY(key, i);
28            if(a[j]==key) return j;
29            i++;
30        } while(a[j]!=NIL && i < M);
31        return -1;
32    }
```

ম্যাপের সাহায্যে স্ট্রিং হ্যাশিং [ইনডেক্স]

```
1    /*
2    hash() takes the string and next free index as parameter.
3    if string is found in map, it returns its index.
4    else, it inserts in current free index and updates the free
5    */
6
7    inline int hash(char *s, int &n) {
8        int ret; it = M.find(s);
9        if(it == M.end()) M.insert(pair< string, int > (s, ret =
10        else ret = it->second;
11        return ret;
12    }
```

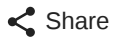## বার্নস্টাইন স্ট্রিং হ্যাশিং [ইনডেক্স]

```
1    /*
2    u64 stands for unsigned long long type. Must use unsigned typ
3    hash = hash * 33 ^ c, reliable hash
4    */
5    u64 hash(const char* s) {
6        u64 hash = 0, c;
7        while((c = *s++)) hash = ((hash << 5) + hash) ^ c;
8        return hash;
9    }
```

Share this:

< Share

★ Like

7 bloggers like this.

[মন্তব্য (0)](#)

1. কোন মন্তব্য নেই এখনও

1. মার্চ 16, 2011; 11:04 পূর্বাহ্ন এ
   [programing learning « LOVE FOR ALL](#)

মন্তব্যসমূহ বন্ধ করা হয়েছে।
[RSS feed](#)

Email Subscription

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Join 88 other followers

Enter your email address

Sign me up!

Calender

অক্টোবর 2017

শনি	রবি	সোম	ম	বুধ	বৃহ.	শু.
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

[« ফেব্রু.](#)

Archives

- [ফেব্রুয়ারি 2011](#) (1)
- [অক্টোবর 2010](#) (1)
- [অগাস্ট 2010](#) (1)
- [জুলাই 2010](#) (2)
- [নভেম্বর 2009](#) (1)

Tags

[avro](#) [bangla](#) [c++](#) [data structure](#) [pair](#) [priority queue](#) [programming](#) [queue](#) [solaiman lipi](#) [stack](#) [stl](#)

Blog Stats

- 37,632 hits

Top Posts

- [কোড লাইব্রেরি](#)
- [অন্যান্য লিঙ্ক](#)
- [C++ STL :: stack](#)
- [C++ STL :: vector](#)

Meta

- [রেজিস্টার](#)
- [লগ ইন](#)
- [আর.এস.এস.](#)
- [মন্তব্য RSS](#)

- [WordPress.com](https://wordpress.com)

[উপরে](#)
[Blog at WordPress.com.](https://wordpress.com)