Team Reference Document

Encore @ Harbin Institute of Technology

June 24, 2013

Contents

1 String Processing 1
- 1.1 AC Automaton 1
- 1.2 Suffix Array 1
- 1.3 Suffix Automaton 2
- 1.4 KMP 2
- 1.5 Algorithm Z 2

2 Network Flow 2
- 2.1 Max flow 2
- 2.2 Cost flow 3

3 Data Structure 3
- 3.1 DLX exact cover 3
- 3.2 DLX fuzzy cover 4
- 3.3 Partition Tree 4
- 3.4 Leftist Tree 4
- 3.5 Cartesian Tree 5
- 3.6 Splay 5

1 String Processing

1.1 AC Automaton

```
#define code(ch) ((ch) - 'A')
const int KIND = 26, MAXN = 3000000;
struct node {
  node* nxt[KIND], *fail;
  int count, id;
} pool[MAXN], *pp, *root, *q[MAXN];
node *newNode() {
  pp->fail = NULL;
  pp->count = 0;
  memset(pp->nxt, 0, sizeof (pp->nxt));
  return pp++;
}
void initialize() {
  pp = pool;
  root = newNode();
}
void insert(const char * str, int id) {
  node * now = root;
  while (*str) {
    int i = code(*str);
    now->nxt[i] = now->nxt[i] == 0 ? newNode() : now->nxt[i];
    now = now->nxt[i];
    str++;
  }
  now->count++, now->id = id;
}
void buildFail(node*& now, int ith) {
  if(now == root) now->nxt[ith]->fail = root;
  node* tmp = now->fail;
  while(tmp) {
    if(tmp->nxt[ith] != NULL) {
      now->nxt[ith]->fail = tmp->nxt[ith];
      return;
    }
    tmp = tmp->fail;
  }
  if(tmp == NULL) now->nxt[ith]->fail = root;
}
void build() {
  int head = 0, tail = 0;
  q[tail++] = root;
  while (head != tail) {
    node * beg = q[head++];
    for (int i = 0; i < KIND; i++) {
      if (beg->nxt[i] == NULL) continue;
      buildFail(beg, i);
      q[tail++] = beg->nxt[i];
    }
  }
}
node* goStatus(node* now, int ith) {
  node * tmp = now;
  while(now->nxt[ith] == NULL && now != root)
    now = now->fail;
  now = now->nxt[ith];
  return now == NULL ? root : now;
}
void query(const char* str) {
  node * p = root, * tmp;
  int tail = 0;
  while (*str) {
    tmp = p = goStatus(p, code(*str));
    while (tmp != root && tmp->count != -1) {
      q[tail++] = tmp;
      tmp->count = -1;
      tmp = tmp->fail;
    }
    str++;
  }
}
```

1.2 Suffix Array

```
const int MAXN = 50001;
int sfx[MAXN], temp[MAXN], key[MAXN][2];
int _rank[MAXN], bucket[MAXN], height[MAXN];
// _rank from 0 to n - 1
void radixSort(int* in, int n, int idx, int* out) {
  memset(bucket, 0, sizeof(int) * (n + 1));
  for (int i = 0; i < n; i++) bucket[key[i][idx]]++;
  for (int i = 1; i <= n; i++) bucket[i] += bucket[i - 1];
  for (int i = n - 1;i >= 0;i--)out[--bucket[key[i][idx]]]=in[i];
}
#define KEY0(i) key[i][0]
#define KEY1(i) key[i][1]
int cmp(int i, int j) {
return KEY0(i) == KEY0(j) ? KEY1(i) < KEY1(j) : KEY0(i) < KEY0(j);
}
/*text can't contain 0, 0 is used as terminal*/
void buildSA(const char* text, int n) {
  for (int i = 0; i < n; i++)
    sfx[i] = i, key[i][0] = text[i], key[i][1] = 0;
  sort(sfx, sfx + n, cmp);
  for(int i = 0;i < n;i++) key[i][0] = text[sfx[i]];
  int wid = 1;
  while (wid < n) {
    _rank[sfx[0]] = 0;
    for (int i = 1; i < n; i++)
      _rank[sfx[i]] = _rank[sfx[i - 1]] + cmp(i - 1, i);
    for (int i = 0; i < n; i++) {
      sfx[i] = i;
      key[i][1] = i + wid < n ? _rank[i + wid]: 0;
    }
    radixSort(sfx, n, 1, temp);
    for(int i = 0;i < n;i++) key[i][0] = _rank[temp[i]];
    radixSort(temp, n, 0, sfx);
    for(int i = 0;i < n;i++) key[i][0] = _rank[sfx[i]];
    for(int i = 0;i < n;i++)
      key[i][1] = wid + sfx[i] < n ? _rank[sfx[i] + wid] : 0;
    wid <<= 1;
  }
}
void calHeight(const char* text, int* _rank, int n) {
  //height[i] = lcp(suffix(sa[i - 1]), suffix[sa[i]])
  for(int i = 0; i < n; i++) _rank[sfx[i]] = i;
  height[0] = 0;
  for(int i = 0, k = 0, j; i < n; i++) {
    if(_rank[i] != 0) {
      if(k > 0) k-- ;
      for (j = sfx[_rank[i] - 1]; text[i + k] == text[j + k]; k++);
      height[_rank[i]] = k;
    }
  }
}
int RMQ[MAXN][20];
//n = len(text),height[0] means nothing
void buildRMQ(int n, int* height) {
  for(int i = 1; i <= n; i++) RMQ[i][0] = height[i - 1];
  for (int j = 1; j <= log(n + 0.00) / log(2.0); j++)
    for (int i = 1; i + (1 << j) - 1 <= n; i++)
      RMQ[i][j] = min(RMQ[i][j - 1],RMQ[i + (1<<(j-1))][j - 1]);
}
int queryRMQ(int a, int b) {
  int len = log(b - a + 1.0) / log(2.0);
  return min(RMQ[a][len], RMQ[b - (1 << len) + 1][len]);
}
int queryLCP(int a, int b) {
  a = _rank[a] + 1, b = _rank[b] + 1;
  if(a > b) swap(a, b);
  return queryRMQ(a + 1, b);
}
```

1.3 Suffix Automaton

```
namespace SAM {
const int MAXN = 600000;
struct Node {
    Node *ch[26], *f; int l;
} a[MAXN], *root, *acc, *ptr;
void Initial() {
    memset(a, 0, sizeof(a));
    acc = root = a, ptr = a + 1;
}
void AddSuffix(int x) {
    using namespace std;
    Node * cur = ptr++, *fail = acc;
    cur->l = acc->l + 1; acc = cur;
    for(;fail && !fail->ch[x];fail = fail->f)
        fail->ch[x] = cur;
    if(!fail) {
        cur->f = root;
    } else if(fail->l + 1 == fail->ch[x]->l) {
        cur->f = fail->ch[x];
    } else {
        Node* r = ptr++, * q = fail->ch[x];
        *r = *q, r->l = fail->l + 1;
        cur->f = q->f = r;
        for(;fail && fail->ch[x] == q;fail = fail->f)
            fail->ch[x] = r;
    }
}
int lcs(const char * src, const char * dest) {
    Initial();
    int n = strlen(src), m = strlen(dest), ans = 0, mid = 0;
    Node * acc = root;
    for(int i = 0;i < n;i++) {
        SAM::AddSuffix(src[i] - ’a’);
    }
    for(int i = 0;i < m;++i) {
        int v = dest[i] - ’a’;
        if(acc->ch[v]) {
            ++mid;
            acc = acc->ch[v];
        } else {
            for(;acc && !acc->ch[v];acc = acc->f);
            mid = acc ? acc->l + 1 : 0;
            acc = acc ? acc->ch[v] : root;
        }
        ans = max(ans, mid);
    }
    return ans;
}
}
```

1.4 KMP

```
//be careful with mod string and main string
void prefix(const char *mode, int *next) {
  int m = strlen(mode), k = -1, i;
  next[0] = -1;
  for (i = 1; i < m; i++) {
    while (k > -1 && mode[k + 1] != mode[i]) k = next[k];
    if (mode[k + 1] == mode[i]) k++;
    next[i] = k;
  }
}
int KMP(const char *main, const char *mode) {
  int n = strlen(main), m = strlen(mode), q = -1, ans = 0;
  int next[LEN], i;
  prefix(mode, next);
  for (i = 0; i < n; i++) {
    while (q > -1 && mode[q + 1] != main[i]) q = next[q];
    if (mode[q + 1] == main[i]) q++;
    if (q == m - 1) {
      ans++;
      q = next[q];
    }
  }
  return ans;
}
```

1.5 Algorithm Z

```
#include <cmath>
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
void get_suffix(const char* sub, int len, int next[]) {
  //extend[i] = len(lcp(sub, sub.substr(i)))
  int pos = 1, j = 0;
  while(sub[j + 1] == sub[j]) j++;
  next[0] = len, next[pos] = j;
  for(int i = 2;i < len;i++) {
    int ll = pos + next[pos], cur = next[i - pos];
    if(ll > i + cur) {
      next[i] = cur;
    } else {
      j = max(ll - i, 0);
      while(sub[i + j] == sub[j] && i + j < len) j++;
      next[i] = j;
      pos = i;
    }
  }
}
void extend_kmp(const char* str, int n, const char* sub, int m,
         int extend[], int next[]) {
  get_suffix(sub, m, next);
  int j = 0, pos = 0;
  while(str[j] == sub[j] && j < n && j < m) j++;
  extend[0] = j;
  for(int i = 1;i < n;i++) {
    int ll = pos + extend[pos], cur = next[i - pos];
    if(ll > i + cur) {
      extend[i] = cur;
    } else {
      j = max(ll - i, 0);
      while(str[i + j] == sub[j] && i + j < n && j < m) j++;
      extend[i] = j;
      pos = i;
    }
  }
}
```

2 Network Flow

2.1 Max flow

```
const int V = 1010;
const int E = V*V*2;
const int INF = 1<<29;
typedef struct Edge{
  int v, cap, flow;
  Edge *next, *re;
}Edge;
class MaxFlow{
public:
  Edge edge[E], *adj[V], *pre[V], *arc[V];
  int e, n, d[V], q[V], numb[V];
  void Init(int x){
    n = x;
    for (int i = 0; i < n; ++i) adj[i] = NULL;
    e = 0;
  }
  void Addedge(int x, int y, int f) {
    edge[e].v = y, edge[e].cap = f, edge[e].next = adj[x], edge[e].
             re = &edge[e+1]; adj[x] = &edge[e++];
    edge[e].v = x, edge[e].cap = 0, edge[e].next = adj[y], edge[e].
             re = &edge[e-1]; adj[y] = &edge[e++];
  }
  void Bfs(int v) {
    int front = 0, rear = 0, r = 0, dis = 0;
    for (int i = 0;i < n; ++i) d[i] = n, numb[i] = 0;
    d[v] = 0;++numb[0];
    q[rear++] = v;
    while (front != rear) {
      if (front == r) ++dis, r = rear;
      v = q[front++];
      for (Edge *i = adj[v];i != NULL;i = i->next) {
        int t = i->v;
        if (d[t] == n) d[t] = dis, q[rear++] = t, ++numb[dis];
      }
    }
  }
  int Maxflow(int s, int t){
    int ret = 0, i, j;
    Bfs(t);
    for (i = 0; i < n; ++i) pre[i] = NULL, arc[i] = adj[i];
    for (i = 0; i < e; ++i) edge[i].flow = edge[i].cap;
    i = s;
    while (d[s] < n) {
      while (arc[i] && (d[i] != d[arc[i]->v]+1 || !arc[i]->flow))
               arc[i] = arc[i]->next;
      if (arc[i]) {
        j = arc[i]->v;
        pre[j] = arc[i];
        i = j;
        if (i == t) {
          int update = INF;
          for (Edge *p = pre[t];p != NULL;p = pre[p->re->v])
                   checkmin(update, p->flow);
          ret += update;
```

```
       for (Edge *p = pre[t];p != NULL;p = pre[p->re->v]) p->flow
              -= update, p->re->flow += update;
       i = s;
      }
    }
    else {
     int min = n - 1;
     for (Edge *p = adj[i];p != NULL;p = p->next) if(p->flow)
            checkmin(min, d[p->v]);
     if (--numb[d[i]] == 0) return ret;
     d[i] = min + 1;
     ++numb[d[i]];
     arc[i] = adj[i];
     if (i != s) i = pre[i]->re->v;
    }
   }
   return ret;
 }
};
```

2.2 Cost flow

```
using namespace std;
typedef long long USETYPE;
const USETYPE INF = numeric_limits<USETYPE>::max();//<limits>
template<typename T = int>
class mincost {
private:
   const static int N = 1000;
   const static int E = 100000;
   struct edge {
      int u, v;
      T cost, cap;
      edge *nxt;
   } pool[E], *g[N], *pp, *pree[N];
   T dist[N];

   bool SPFA(int n,int s, int t) {
      fill(dist, dist + n, INF);
      int tail = 0, q[N] = {s};
      dist[s] = 0;
      bool vst[N] = {false};
      vst[s] = true;
      for(int i = 0; i <= tail; i++) {
         int u = q[i % n];
         for(edge *j = g[u]; j != NULL; j= j->nxt) {
            int v = j->v;
            if(j->cap && dist[u] != INF && dist[v] > dist[u] + j->
                   cost) {
               dist[v] = dist[u] + j->cost;
               pree[v] = j;
               if(!vst[v]) {
                  tail++;
                  q[tail % n] = v;
                  vst[v] = true;
               }
            }
         }
         vst[u] = false;
      }
      return dist[t] < INF;
   }
public:
#define OP(i) (((i) - pool) ^ 1)
   void addedge(int u, int v, T cap, T cost) {
      pp->u = u, pp->v = v;
      pp->cost = cost, pp->cap = cap;
      pp->nxt = g[u],g[u] = pp++;
   }
   void initialize() {
      CC(g, 0);
      pp = pool;
   }
   pair<T, T> mincostflow(int n, int s, int t) {
      T flow = 0, cost = 0;
      while(SPFA(n, s, t)) {
         T minf = INF;
         for(int i = t; i != s; i = pree[i]->u)
            minf = min(minf, pree[i]->cap);
         for(int i = t; i != s; i = pree[i]->u) {
            pree[i]->cap -= minf;
            pool[OP(pree[i])].cap += minf;
            cost += minf * pree[i]->cost;
         }
         flow += minf;
      }
      return make_pair(flow, cost);
   }
};
```

3 Data Structure

3.1 DLX exact cover

```
const int SIZE = 16, SQRTSIZE = 4;//here
const int ALLSIZE = SIZE * SIZE, ROW = SIZE * SIZE * SIZE;
const int INF = 100000000, COL = SIZE * SIZE * 4;
const int N = ROW * COL, HEAD = 0;
#define BLOCK(r, c) ((r) * SQRTSIZE + c)
#define CROW(r, c, k) ((r) + (c) * SIZE + (k) * SIZE * SIZE)
#define ROWCOL(i, j) ((i) * SIZE + (j))
#define ROWCOLOR(i, k) (ALLSIZE + (i) * SIZE + k)
#define COLCOLOR(j, k) (2 * ALLSIZE + (j) * SIZE + k)
#define BLOCKCOLOR(i, j, k) (3*ALLSIZE+BLOCK((i/SQRTSIZE),(j/
       SQRTSIZE))*SIZE+(k))
int maps[ROW][COL], ans[N];
char sudoku[SIZE][SIZE];
int r[N], l[N], u[N], d[N], c[N], s[N];
int n, m, ansd, row[N];
void resume(const int col) {
   for (int i = u[col]; i != col; i = u[i]) {
      for (int j = l[i]; j != i; j = l[j]) {
         u[d[j]] = j;
         d[u[j]] = j;
         s[c[j]]++;
      }
   }
   r[l[col]] = col;
   l[r[col]] = col;
}
void cover(const int col) {
   r[l[col]] = r[col];
   l[r[col]] = l[col];
   for (int i = d[col]; i != col; i = d[i]) {
      for (int j = r[i]; j != i; j = r[j]) {
         u[d[j]] = u[j];
         d[u[j]] = d[j];
         s[c[j]]--;
      }
   }
}
void initialize(int n, int m) {
   l[HEAD] = m;
   r[HEAD] = 1;
   for (int i = 1; i <= m; i++) {
      if (i == m) {
         r[i] = HEAD;
      } else {
         r[i] = i + 1;
      }
      l[i] = i - 1;
      c[i] = u[i] = d[i] = i;
      s[i] = 0;
   }
   int size = m;
   for (int i = 1; i <= n; i++) {
      int first = 0;
      for (int j = 1; j <= m; j++) {
         if (maps[i - 1][j - 1] == 0) continue;
         size++;
         int tmp = u[j];
         u[j] = size; d[tmp] = size;
         d[size] = j; u[size] = tmp;
         if (!first) {
            first = size;
            l[size] = r[size] = size;
         } else {
            tmp = l[first];
            r[tmp] = size;
            l[size] = tmp;
            l[first] = size;
            r[size] = first;
         }
         row[size] = i;
         s[j]++;
         c[size] = j;
      }
   }
}
bool dfs(int depth) {
   if (r[HEAD] == HEAD) {
      ansd = depth;
      return true;
   }
   int minn = INF, v;
   for (int i = r[HEAD]; i != HEAD; i = r[i]) {
      if (s[i] < minn) {
         v = i;
         minn = s[i];
      }
   }
   cover(v);
   for (int i = d[v]; i != v; i = d[i]) {
      for (int j = r[i]; j != i; j = r[j])
         cover(c[j]);
      ans[depth] = row[i] - 1;
      if (dfs(depth + 1))
         return true;
      for (int j = l[i]; j != i; j = l[j])
         resume(c[j]);
   }
   resume(v);
   ans[depth] = -1;
   return false;
}

int main() {
   n = ROW;
   m = COL;
   while (scanf(" %c", &sudoku[0][0]) == 1) {
      for(int i = 0; i < SIZE; i++) {
         for(int j = 0; j < SIZE; j++) {
            if(i + j) scanf(" %c", &sudoku[i][j]);
         }
      }
      memset(maps, 0, sizeof (maps));
      for (int i = 0; i < SIZE; i++) {
         for (int j = 0; j < SIZE; j++) {
            if (sudoku[i][j] == ’-’) {
               for (int k = 0; k < SIZE; k++) {
                  maps[CROW(i, j, k)][ROWCOL(i, j)] = 1;
                  maps[CROW(i, j, k)][ROWCOLOR(i, k)] = 1;
                  maps[CROW(i, j, k)][COLCOLOR(j, k)] = 1;
                  maps[CROW(i, j, k)][BLOCKCOLOR(i, j, k)] = 1;
               }
            } else {
               int k = sudoku[i][j] - ’A’;//here
               maps[CROW(i, j, k)][ROWCOL(i, j)] = 1;
               maps[CROW(i, j, k)][ROWCOLOR(i, k)] = 1;
               maps[CROW(i, j, k)][COLCOLOR(j, k)] = 1;
               maps[CROW(i, j, k)][BLOCKCOLOR(i, j, k)] = 1;
            }
         }
         initialize(n, m);
      }
      if (dfs(0)) {
         for (int i = 0; i < ansd; i++)
            sudoku[ans[i] % SIZE][ans[i] % ALLSIZE / SIZE] = ans[i
                  ] / ALLSIZE + 'A';//here
         for(int i = 0; i < SIZE; i++) {
            for(int j = 0; j < SIZE; j++)
               putchar(sudoku[i][j]);
            puts("");
         }
      }
      puts("");
   }
   return 0;
}
```

3.2 DLX fuzzy cover

```
const int ROW = 56;
const int COL = 56;
const int N = ROW * COL, HEAD = 0;
const int INF = 1000000000;
int maps[ROW][COL], ansq[ROW], row[N];
int s[COL], u[N], d[N], l[N], r[N], c[N];
void build(int n, int m) {
   r[HEAD] = 1;
   l[HEAD] = m;
   for (int i = 1; i <= m; i++) {
      l[i] = i - 1;
      r[i] = (i + 1) % (m + 1);
      c[i] = d[i] = u[i] = i;
      s[i] = 0;
   }
   int size = m;
   for (int i = 1; i <= n; i++) {
      int first = 0;
      for (int j = 1; j <= m; j++) {
         if (!maps[i - 1][j - 1]) continue;
         size++;
         d[u[j]] = size;
         u[size] = u[j];
         d[size] = j;
         u[j] = size;
         if (!first) {
            first = size;
            l[size] = size;
            r[size] = size;
         } else {
            l[size] = l[first];
            r[size] = first;
            r[l[first]] = size;
            l[first] = size;
         }
         c[size] = j;
         s[j]++;
      }
   }
}
inline void coverc(int col) {
   for(int i = d[col]; i != col; i = d[i]) {
      r[l[i]] = r[i];
      l[r[i]] = l[i];
   }
}
inline void resumec(int col) {
   for(int i = u[col]; i != col; i = u[i]) {
      l[r[i]] = i;
      r[l[i]] = i;
   }
}
bool vis[COL];
int H() {
   int cnt = 0;
   memset(vis,0,sizeof(vis));
   for (int i = r[HEAD]; i != HEAD; i = r[i]) {
      if (vis[i]) continue;
      cnt++;
      vis[i] = 1;
      for (int j = d[i]; j != i; j = d[j])
         for (int k = r[j]; k != j; k = r[k])
            vis[c[k]] = 1;
   }
   return cnt;
}
int cut,nextcut;
bool dfs(int dep) {
   if (!r[HEAD]) return true;
   int now, minn = ROW;
   for (int i = r[HEAD]; i != HEAD; i = r[i])
      if (minn > s[i]) {
         minn = s[i];
         now = i;
      }
   for (int j = d[now]; j != now; j = d[j]) {
      //ansq[dep]=row[rp];
      coverc(j);
      for (int i = r[j]; i != j; i = r[i])
         coverc(i);
      int tmp = dep + 1 + H();
      if(tmp > cut) nextcut = min(tmp, nextcut);
      else if (dfs(dep + 1)) return true;
      for (int i = l[j]; i != j; i = l[i])
         resumec(i);
      resumec(j);
   }
   return false;
}
int IDAstar(int n) {
   cut = H();
   nextcut = n;
   memset(vis,0,sizeof(vis));
   while(!dfs(HEAD)) {
      cut = nextcut;
      nextcut = n;
   }
   return cut;
}
```

3.3 Partition Tree

```
/* NlogN find Kth number in any interval */
class partition_tree {
private:
   static const int N = 100005;
   static const int DEPTH = 20;
   int tree[DEPTH][N * 4], sorted[N];
   int toleft[DEPTH][N * 4];
   int n;
public:
   void initialize(int n, int *array) {
      this->n = n;
      for (int i = 1; i <= n; i++)
         sorted[i] = tree[0][i] = array[i];
      sort(sorted + 1, sorted + n + 1);
   }
   void build(int l, int r, int depth) {
      if (l == r) return;
      int mid = (l + r) / 2, same = 0, less = 0;
      for (int i = l; i <= r; i++)
         less += (tree[depth][i] < sorted[mid]);
      same = mid - l + 1 - less;
      int lpos = l, rpos = mid + 1;
      for (int i = l; i <= r; i++) {
         int w = tree[depth][i];
         if (w < sorted[mid]) tree[depth + 1][lpos++] = w;
         else if (w == sorted[mid] && same) {
            tree[depth + 1][lpos++] = w;
            same--;
         }
         else
            tree[depth + 1][rpos++] = w;
         toleft[depth][i] = toleft[depth][l - 1] + lpos - l;
      }
      build(l, mid, depth + 1);
      build(mid + 1, r, depth + 1);
   }
// ptree.query(1, n, a, b, 0, k) th kth number of [a, b]
   int query(int L, int R, int l, int r, int depth, int k) {
      if (l == r) return tree[depth][l];
      int cnt, mid = (R + L) / 2, tmpl, tmpr;
      cnt = toleft[depth][r] - toleft[depth][l - 1];
      if (cnt >= k) {
         tmpl = L + toleft[depth][l - 1] - toleft[depth][L - 1];
         tmpr = tmpl + cnt - 1;
         return query(L, mid, tmpl, tmpr, depth + 1, k);
      } else {
         tmpr = r + toleft[depth][R] - toleft[depth][r];
         tmpl = tmpr - (r - l - cnt);
         return query(mid + 1, R, tmpl, tmpr, depth + 1, k - cnt);
      }
   }
};
```

3.4 Leftist Tree

```
#define CMP(a, b) ((a) > (b))
#define DIST(v) ((v == NULL) ? -1 : (v->dist))
//use it template carefully
template<typename T>
class leftist_tree {
private:
   class node {
   public:
      T v;
      int dist;
      node *rr, *ll;
      node(){rr = ll = NULL; dist = 0;}
      node(T v){this->v = v; rr = ll = NULL;dist = 0;}
   };
   node* root;
   int s;
   node* merge(node* &left, node* &right) {
      if(left == NULL) return right;
      if(right == NULL) return left;
      if(CMP(right->v, left->v)) swap(left, right);
      left->rr = merge(left->rr, right);
      if(DIST(left->rr)>DIST(left->ll))swap(left->ll, left->rr);
      left->dist = DIST(left->rr) + 1;
      return left;
   }
   void clear(node* root) {
      if(root == NULL) return;
      clear(root->ll);
      clear(root->rr);
      delete root;
      root = NULL;
   }
public:
   leftist_tree(){root = NULL;s = 0;}
   ~leftist_tree(){clear(root);}
   void push(T v) {
      node * newNode = new node(v);
      root = merge(newNode, root);
      s++;
   }
   void clear(){clear(root);}
   int size(){return this->s;}
   T top(){return root->v;}
   void pop() {
      node *tmp = root;
      root = merge(root->ll, root->rr);
      delete tmp;
      s--;
   }
   void merge(leftist_tree<T>& tree) {
      this->root = merge(root, tree.root);
      s += tree.s;
      tree.root = NULL;
   }
   void makeNULL(){root = NULL;}
};
```

3.5 Cartesian Tree

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <cstring>
using namespace std;
const int N = 100000;
struct node {
 int key, value, id;
 bool operator < (const node& oth) const {
  return key < oth.key;
 }
}nodes[N];
/*lt[i] is nodes[i]'s left son, shouldn't sort again*/
int lt[N], rt[N], parent[N];
void rotate(int i) {
 while(parent[i]!=-1&&nodes[i].value<nodes[parent[i]].value) {
  rt[parent[i]] = lt[i];
  if(lt[i] != -1) parent[lt[i]] = parent[i];
  lt[i] = parent[i];
  int ff = parent[parent[i]];
  if(ff != -1) {
   parent[i] == lt[ff] ? lt[ff] = i : rt[ff] = i;
  }
  parent[i] = ff;
  parent[lt[i]] = i;
 }
}
int key[N], value[N], pos[N];
void build(int n) {
 sort(nodes, nodes + n);
 int rightmost = 0;
 for(int i = 1;i < n;i++) {
  pos[nodes[i].id] = i;
  rt[rightmost] = i;
  parent[i] = rightmost;
  rightmost = i;
  rotate(i);
 }
}
#define V(i) (i == -1 ? 0 : nodes[i].id + 1)
int main() {
 int n;
 while(scanf("%d", &n) == 1) {
  for(int i = 0;i < n;i++) {
   scanf("%d %d", &nodes[i].key, &nodes[i].value);
   nodes[i].id = i;
   key[i] = nodes[i].key;
   value[i] = nodes[i].value;
   lt[i] = rt[i] = parent[i] = -1;
  }
  build(n);
  printf("YES\n");
  for(int i = 0;i < n;i++) {
   printf("%d %d %d\n", V(parent[pos[i]]),
    V(lt[pos[i]]), V(rt[pos[i]]));
  }
 }
 return 0;
}
```

3.6 Splay

```
struct node {
/* virtual node if tot is equal to 0*/
#define __JUDGE if(tot == 0) return;
 static const int INF = 100000000;
 node* ch[2], *pre;
 int v, minn, tot, delta, flip;
 node(int v, int tot, node* l, node* r, node* pre)
  : pre(pre), v(v), minn (v), tot(tot), delta(0), flip(0) {
  ch[0] = l, ch[1] = r;
 }
 inline int min_v() { return minn; }
 inline int size() { return tot; }
 void reverse() { __JUDGE flip ^= 1; }
 void add(int d) { __JUDGE minn += d, delta += d, v += d; }
 void push_down() {
  __JUDGE
  if(delta) {
   if(ch[0]->tot) ch[0]->add(delta);
   if(ch[1]->tot) ch[1]->add(delta);
  }
  if(flip) {
   swap(ch[0], ch[1]);
   if(ch[0]->tot) ch[0]->reverse();
   if(ch[1]->tot) ch[1]->reverse();
  }
  flip = delta = 0;
 }
 void push_up() {
  __JUDGE
  tot = ch[0]->size() + ch[1]->size() + 1;
  minn = min(v, min(ch[0]->min_v(), ch[1]->min_v()));
 }
};
class splay_tree {
public:
 splay_tree() {
  null = new node(node::INF, 0, 0, 0, 0);
  root = null;
 }
 ~splay_tree() {
  clear(root);
  delete null;
 }
 // make a sequence from 1 to n do build(0, n + 1, val)
 // and make sure val[0] = va[1] = INF;
 void build(int l, int r, int* val) {
  if(l > r) return;
  build(l, r, root, null, val);
 }
#define centre (root->ch[1]->ch[0])
 int min_value(int a, int b) {
  makeInterval(a, b);
  return centre->min_v();
 }
 void add_value(int a, int b, int value) {
  makeInterval(a, b);
  centre->add(value);
  splay(centre, null);
 }
 void reverse(int a, int b) {
  if(a == b) return;
  makeInterval(a, b);
  centre->reverse();
  splay(centre, null);
 }
 void revolve(int a, int b, int c) { // c < b - a + 1
  if(c == 0) return;
  int len = b - a + 1;
  reverse(a, a + len - c - 1);
  reverse(a + len - c, b);
  reverse(a, b);
 }
 void insert(int a, int c) {
  makeInterval(a + 1, a);
  centre = new node(c, 1, null, null, root->ch[1]);
  root->ch[1]->push_up();
  root->push_up();
  splay(centre, null);
 }
 void erase(int a) {
  makeInterval(a, a);
  delete centre;
  centre = null;
  root->ch[1]->push_up();
  root->ch[0]->push_up();
 }
#undef centre
 void clear() { clear(root); }
private:
 node* root, * null;
 void clear(node*& now) {
  if(now == null) return;
  clear(now->ch[0]);
  clear(now->ch[1]);
  delete now;
  now = null;
 }
 /* 0: right rotate, 1: left rotate*/
 void rotate(node* x, int type) {
  node *y = x->pre;
  y->push_down(), x->push_down();
  y->ch[!type] = x->ch[type];
  if (x->ch[type] != null)
   x->ch[type]->pre = y;
  x->pre = y->pre;
  if (y->pre != null) {
   if(y->pre->ch[1] == y)
    y->pre->ch[1] = x;
   else
    y->pre->ch[0] = x;
  }
  x->ch[type] = y, y->pre = x;
  if (y == root) root = x;
  y->push_up(), x->push_up();
 }
 void splay(node* x, node* f) {
  x->push_down();
  while(x->pre != f) {
   if (x->pre->pre == f) {
    if (x->pre->ch[0] == x)
     rotate(x, 1);
    else
     rotate(x, 0);
   } else {
    node *y = x->pre;
    node *z = y->pre;
    if (z->ch[0] == y) {
     if (y->ch[0] == x) // 1
      rotate(y, 1), rotate(x, 1);
     else // z
      rotate(x, 0), rotate(x, 1);
    } else {
     if (y->ch[1] == x) // 1
      rotate(y, 0), rotate(x, 0);
     else // z
      rotate(x, 1), rotate(x, 0);
    }
   }
  }
  x->push_up();
 }
 void build(int l, int r, node*& now, node* pre, int* val) {
  if(l > r) return;
  int mid = (l + r) / 2;
  now = new node(val[mid], 1, null, null, pre);
  build(l, mid - 1, now->ch[0], now, val);
  build(mid + 1, r, now->ch[1], now, val);
  now->push_up();
 }
 // the flag node is !not! included, be careful when make interval
 void findK(int k, node* pre) {
  node* now = root;
  while(true) {
   now->push_down();
   int s = now->ch[0]->size();
   if(s == k) break;
   else if(s > k)
    now = now->ch[0];
   else {
    now = now->ch[1];
    k -= s + 1;
   }
  }
  splay(now, pre);
 }
 void makeInterval(int a, int b) {
    findK(a - 1, null);
    findK(b + 1, root);
  }
}tree;
const int N = 300000;
int val[N], n, m, a, b, c;
int main() {
  char cmd[100];
  while(scanf("%d", &n) == 1) {
    for(int i = 1;i <= n;i++) scanf("%d", &val[i]);
    val[0] = val[n + 1] = node::INF;
    tree.clear();
    tree.build(0, n + 1, val);
    scanf("%d", &m);
    REP(i, 0, m) {
      scanf("%s", cmd);
      if(!strcmp(cmd, "ADD")) {
        scanf("%d %d %d", &a, &b, &c);
        tree.add_value(a, b, c);
      } else if(!strcmp(cmd, "REVERSE")) {
        scanf("%d %d", &a, &b);
        tree.reverse(a, b);
      } else if(!strcmp(cmd, "REVOLVE")) {
        scanf("%d %d %d", &a, &b, &c);
        int tot = b - a + 1;
        c = (c % tot + tot) % tot;
        tree.revolve(a, b, c);
      } else if(!strcmp(cmd, "INSERT")) {
        scanf("%d %d", &a, &c);
        tree.insert(a, c);
      } else if(!strcmp(cmd, "DELETE")) {
        scanf("%d", &a);
        tree.erase(a);
      } else if(!strcmp(cmd, "MIN")) {
        scanf("%d %d", &a, &b);
        printf("%d\n", tree.min_value(a, b));
      }
    }
  }
  return 0;
}
```