Pontificia Universidad Católica del Perú - FCI

XieXieLucas Notebook - Froz/Phibrain/Ands

November 9, 2017

Contents

1 Centroid Decomposition

```
#define N 100002

inline ll ma(ll a, ll b){ return ((a-b>0)? a:b);}
inline ll mi(ll a, ll b){return ((a-b>0)? b:a);}

struct CD{
vector< int > graph[N];
int sub[N],p[N];
//sub[i]: size del nodo i luego de descomponer el tree
//p[i]: padre del nodo i luego de descomponer el tree
//notar que el padre del centroid es -2
// el tree esó 1 0 base
//para inicializar addEddge(a,b);
//para construir el centroid tree, solo llamar init(root); root:
    root del tree
void addEdge(int &a, int &b){
       graph[a].pb(b);
       graph[b].pb(a);
}
inline void dfs(int cur, int parent){
       sub[cur] = 1;
       for(int i = 0; i < sz(graph[cur]); ++i){
              int to = graph[cur][i];
              if(to != parent && p[to] == -1){
                     dfs(to, cur);
                     sub[cur] += sub[to];
              }
       }
}
inline void decompose(int cur, int parent, int sb, int prevc){
       for(int i = 0; i < sz(graph[cur]); ++i){
              int to = graph[cur][i];
              if(to != parent && p[to] == -1 && (2 * sub[to] > sb)
                  ){
                     decompose(to, cur, sb, prevc);
                     return;
              }
       }
       p[cur] = prevc;
       for(int i = 0; i < sz(graph[cur]); ++i){
```

```
                        int to = graph[cur][i];
                        if(p[to] == -1){
                                dfs(to, - 1);
                                decompose(to, cur, sub[to], cur);
                        }
                }
        }
        inline void init(int start){
                for(int i = 0; i < N; ++i) p[i] = -1;
                dfs(start, - 1);
                decompose(start, -1, sub[start], -2);
        }
};
int cnt=1;
vi adj[N];
int d[N];
inline void make(int &u, int x, int depth){
        d[u]=depth;
        for(auto v :adj[u]) if(v!=x) make(v,u,depth+1);
}
int main() {
        fastio;
        int n; cin>>n;
        CD cd; //cd.n=n;
        REP(i,0,n-1) {
                ll a,b; cin>>a>>b;
                cd.addEdge(a,b);
        }
        cd.init(1);
        int pa, root;
        REP(i,1,n+1) {
                pa=cd.p[i];
                if(pa==-2) root=i;
                if(pa!=-2) {
                        adj[i].pb(pa);
                        adj[pa].pb(i);
                }
        }
        make(root,0,1);
        char is;map<int, string> m;int k=1,flag=1;
        for(is='A'; is<='Z'; is++) m[k++]=is;
        REP(i,1,n+1) if(d[i]>26) flag=0;
        if(flag==0) cout<<"Impossible!"<<endl;
        if(flag==1) {
                REP(i,1,n+1) cout<<m[d[i]]<<endl;
        }
        return 0;
}
```

2 Closest Pair

```
//Closest Pair Algorithm with Sweep
//Complexity: O(nlogn)

#define MAX_N 100000
#define px second
#define py first
typedef pair<long long, long long> point;

int N;
point P[MAX_N];
set<point> box;

bool compare_x(point a, point b){ return a.px<b.px; }

inline double dist(point a, point b){
        return sqrt((a.px-b.px)*(a.px-b.px)+(a.py-b.py)*(a.py-b.py));
}
double closest_pair(){
        if(N<=1) return -1;
        sort(P,P+N,compare_x);
        double ret = dist(P[0],P[1]);
        box.insert(P[0]);
        set<point> :: iterator it;
        for(int i = 1,left = 0;i<N;++i){
                while(left<i && P[i].px-P[left].px>ret) box.erase(P[left
                    ++]);
                for(it = box.lower_bound(make_pair(P[i].py-ret,P[i].px-ret)
                    );
                it!=box.end() && P[i].py+ret>=(*it).py;++it)
                ret = min(ret, dist(P[i],*it));
                box.insert(P[i]);
        }
        return ret;
}
```

3 Convex Hull Trick

```
 // Simple Hull
struct HullSimple { // Upper envelope for Maximum.
    // Special case: strictly increasing slope in insertions,
    // increasing value in queries.
    deque<pair<ll, ll> > dq;
    ld cross(pair<ll, ll> l1, pair<ll, ll> l2){
        return (ld)(l2.snd - l1.snd) / (ld)(l1.fst - l2.fst);
    }
    void insert_line(ll m, ll b){
        pair<ll,ll> line = mp(m,b);
        while (sz(dq) > 1 && cross(line, dq[sz(dq)-1]) <=
               cross(dq[sz(dq)-1], dq[sz(dq)-2])) dq.pop_back();
        dq.pb(mp(m,b));
    }

    ll eval(pair<ll, ll> line, ll x){
        return line.fst * x + line.snd;
    }

    ll eval(ll x){
        while (sz(dq) > 1 && eval(dq[0], x) < eval(dq[1],x))
            dq.pop_front();
        return eval(dq[0],x);
    }
};

// Dynamic Hull
// Compile with g++ -std=c++11 file.cpp -o file
typedef long double ld;
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
// Upper envelope for Maximum
struct HullDynamic : public multiset<Line> {
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >=
                                    (y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0: &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

4 Dates

```
//
// Time - Leap years
//

// A[i] has the accumulated number of days from months previous to i
const int A
    [13] = { 0, 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334 };
// same as A, but for a leap year
const int B
    [13] = { 0, 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
// returns number of leap years up to, and including, y
int leap_years(int y) { return y / 4 - y / 100 + y / 400; }
bool is_leap(int y) { return y % 400 == 0 || (y % 4 == 0 && y % 100 != 0)
    ; }
// number of days in blocks of years
```

```
const int p400 = 400*365 + leap_years(400);
const int p100 = 100*365 + leap_years(100);
const int p4  = 4*365 + 1;
const int p1  = 365;
int date_to_days(int d, int m, int y)
{
  return (y - 1) * 365 + leap_years(y - 1) + (is_leap(y) ? B[m] : A[m]) +
      d;
}
void days_to_date(int days, int &d, int &m, int &y)
{
  bool top100; // are we in the top 100 years of a 400 block?
  bool top4;   // are we in the top 4 years of a 100 block?
  bool top1;   // are we in the top year of a 4 block?

  y = 1;
  top100 = top4 = top1 = false;

  y += ((days-1) / p400) * 400;
  d = (days-1) % p400 + 1;

  if (d > p100*3) top100 = true, d -= 3*p100, y += 300;
  else y += ((d-1) / p100) * 100, d = (d-1) % p100 + 1;

  if (d > p4*24) top4 = true, d -= 24*p4, y += 24*4;
  else y += ((d-1) / p4) * 4, d = (d-1) % p4 + 1;

  if (d > p1*3) top1 = true, d -= p1*3, y += 3;
  else y += (d-1) / p1, d = (d-1) % p1 + 1;

  const int *ac = top1 && (!top4 || top100) ? B : A;
  for (m = 1; m < 12; ++m) if (d <= ac[m + 1]) break;
  d -= ac[m];
}
```

5 Divide and Conquer Trick

```
// Divide and Conquer DP optimization.
// Problem: dp[i][j] = min{k>j} (func(j,k) + dp[i-1][k]).
// (That is, split n objects into k buckets with cost
// func per bucket). Necessary condition: argmin(dp[i][j]) <=
// argmin(dp[i][j+1]) (this is "opt")
// Naive complexity: O(kn^2)
// Improved complexity: O(knlog(n))
// Consider checking if opt[i+1][j] <= opt[i][j] <= opt[i][j+1]
// and using a knuth-like O(n^2) loop

const ll INF = 1e18;
int n, k;

ll c[8100];
ll s[8100];
ll dp[810][8100];

ll func(int i, int j){ return (s[j] - s[i])*(j-i); }

void go(int i, int l, int r, int optl, int optr){
        if (l >= r) return;
        int m = (l+r)/2;
        int opt = n;
        dp[i][m] = INF;
        for(int u = optr; u>= optl; u--){
                ll curr = dp[i-1][u] + func(m,u);
                if(curr < dp[i][m]){
                        dp[i][m] = curr;
                        opt = u;
                }
        }
        go(i,l,m,optl, opt);
        go(i,m+1,r,opt,optr);
}

int main(){
        fastio;
        cin >> n >> k;
        REP(i,0,n) cin >> c[i];
        s[0] = 0;
        REP(i,0,n+1) s[i] = s[i-1] + c[i-1];
        REP(i,1,k+1) dp[i][n] = INF;
        REP(i,0,n) dp[0][i] = INF;
        dp[0][n] = 0;
        REP(i,1,k+1) go(i,0,n,0,n);
        cout << dp[k][0] << endl;
        return 0;
}

//Divide and Conquer Trick by Ands
```

```
void compute(int cnt, int l, int r, int optl, int optr){
	if(l > r) return ;
	int mid = ( l + r ) >> 1 ;
	int opt = -1 ;
	ll value = 1e18 ;
	int last = cnt^1 ;
	for(int idx = optl ; idx <= min(mid-1,optr); ++idx){
		ll tmp = dp[last][idx] + C[idx][mid] ;
		if(tmp < value){
			value = tmp ;
			opt = idx ;
		}
	}
	dp[cnt&1][mid] = value ;
	compute(cnt, l, mid-1, optl, opt);
	compute(cnt, mid+1, r, opt, optr);
}


int main(){
	//casos base
	for(int cnt = 2; cnt <= m ; ++cnt) compute(cnt&1, 0, n-1, 0, n-1) ;
}
```

6 Fractions

```
struct Frac{
    int num, den;
    Frac(){
        num = 0; den = 1;
    }
    Frac(int a, int b): num(a), den(b){}
    Frac(int a):num(a), den(1){}

    void normalize(){
        if(num == 0){
            den = 1;
        }
        if(den < 0){
            den = -den;
            num = -num;
        }
    }

Frac fix(int a, int b){
    if(!a) return Frac(0,1);
    if(!b) return Frac(oo,1);
    int foo = gcd(abs(a),abs(b));
    Frac ret = Frac(a/foo, b/foo);
    ret.normalize();
    return ret;
}

Frac operator + (const Frac& other){
    int num2 = num*other.den + den*other.num, den2 = den*other.den;
    return fix(num2,den2);
}

Frac operator - (const Frac& other){
    int num2 = num*other.den - den*other.num, den2 = den*other.den;
    return fix(num2,den2);
}

Frac operator * (int c){
    int num2 = num*c, den2 = den;
    return fix(num2,den2);
}

Frac operator * (const Frac& other){
    int num2 = num*other.num, den2 = den*other.den;
    return fix(num2,den2);
}

Frac operator / (int c){
    int num2 = num, den2 = den * c;
    return fix(num2,den2);
}

Frac operator / (const Frac& other){
    int num2 = num*other.den, den2 = den*other.num;
    return fix(num2,den2);
}

bool operator < (const Frac& other) const{
    if(num * other.den < other.num*den) return true;
    return false;
```

```
    }

    bool operator == (const Frac& other) const{
        if(num == other.num && den == other.den) return true;
        return false;
    }
};
```

7 Longest Increasing Subsequence

```
// Simple O( nlogn ) Longest Increasing Subsequence
// Answer is stored in array b[N]

int LIS( vi &a ){
    int b[N];
    int sz = 0;
    REP(i,0,a.size()){
        int j = lower_bound( b , b + sz , a[ i ] ) - b;
        // (lower) a < b < c
        // (upper) a <= b <= c
        b[ j ] = a[ i ];
        if( j == sz ) sz++;
    }
    return sz;
}
```

8 Matrix Structure

```
const int MN = 111;
const int mod = 10000;

struct matrix {
  int r, c;
  int m[MN][MN];

  matrix (int _r, int _c) : r (_r), c (_c) {
    memset(m, 0, sizeof m);
  }
  void print() {
    for (int i = 0; i < r; ++i) {
      for (int j = 0; j < c; ++j)
        cout << m[i][j] << " ";
      cout << endl;
    }
  }

  int x[MN][MN];
  matrix & operator *= (const matrix &o) {
    memset(x, 0, sizeof x);
    for (int i = 0; i < r; ++i)
      for (int k = 0; k < c; ++k)
        if (m[i][k] != 0)
          for (int j = 0; j < c; ++j) {
            x[i][j] = (x[i][j] + ((m[i][k] * o.m[k][j]) % mod) ) % mod;
          }
    memcpy(m, x, sizeof(m));
    return *this;
  }
};

void matrix_pow(matrix b, long long e, matrix &res) {
  memset(res.m, 0, sizeof res.m);
  for (int i = 0; i < b.r; ++i)
    res.m[i][i] = 1;

  if (e == 0) return;
  while (true) {
    if (e & 1) res *= b;
    if ((e >>= 1) == 0) break;
    b *= b;
  }
}
```

9 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef
tree<
```

```
        int,
        null_type,
        less<int>,
        rb_tree_tag,
        tree_order_statistics_node_update
>ordered_set;
// ordered_set
// X.find_by_order(k) returns an iterator to the k-th largest element (
    counting from zero)
// X.order_of_key(v) returns the number of items in a set that are
    strictly smaller than v
int main() {
        int N;
        ordered_set Y;
        Y.insert(5);
        trace (*Y.find_by_order(0));
}
```

10 Parallel Binary Search

```
//Cada query esta en (low[i], high[i]]
//Tocheck tiene los valores acutales a verificar
//en el bsearch

//Solved puede tener 1, -1
//1: el unico valor posible cumple
//-1: no hay respuesta

int low[MAXN];
int high[MAXN];
char solved[MAXN];
vector< int > tocheck[MAXN];

int main(){
        // Leer n, m
        // Leer a[i], b[i] (i en [1, m])
        // Leer q: queries
        // Leer x[i], y[i], z[i] (i en [0, q])

        for(int i = 0; i < q; ++i)
        low[i] = 0, high[i] = m;
        bool done = 0;
        DSU uf(n); // DSU structure
        int curvis;
        while(!done){
                done = 1;
                for(int i = 0; i < q; ++i){
                        int mid = (low[i] + high[i]) >> 1;
                        tocheck[mid].pb(i);
                }
                uf.clear(n);
                int last = -1;
                for(int value = 0; value <= m; ++value){
                        if(tocheck[value].empty()) continue;
                        for(int i = last + 1; i <= value; ++i)
                                uf.join(a[i], b[i]);
                        last = value;
                        while(!tocheck[value].empty()){
                                int id = tocheck[value].back();
                                tocheck[value].pop_back();
                                int u = x[id], v = y[id];
                                int visited = z[id];
                                if(low[id] + 1 == high[id]) solved[id] = 1;
                                if(uf.connected(u, v)) curvis = uf.size(u);
                                else curvis = uf.size(u) + uf.size(v);
                                if(curvis >= visited) high[id] = value;
                                else low[id] = value;
                                if(low[id] == high[id]) solved[id] = -1;
                        }
                }
                for(int i = 0; i < q; ++i)
                        if(solved[i] == 0) done = 0;
        }
        for(int i = 0; i < q; ++i)
                if(solved[i] == -1) cout << -1 << endl;
                else cout << high[i] << endl;
}
```

11 Unordered Map

```
unordered_map<int,int> mp;
mp.reserve(1024); // power of 2 is better
mp.max_load_factor(0.25); // 0.75 used in java
```