COMP4128 templates

Segment trees

What it does:
For each node in segment tree, it will dictate the range of the original array a[l…r]
Where can we use such data structure?
Segment trees can maintain the sum / min / max / product / xor / or / and for ranges.
It can also maintain the “divide and conquer” result of ranges. (When countering these problems, we need to think carefully about what states we are trying to maintain, how do we merge the states for [l , m] and [m + 1 , r] into [l , r] , and always remember to return struct when query)

The following are some examples:

Segment tree 记忆化分治
SPOj-MULTQ3
Problem statement :
There are N numbers a[0],a[1]..a[N - 1]. Initally all are 0. You have to perform two types of operations :
1) Increase the numbers between indices A and B (inclusive) by 1. This is represented by the command "0 A B"
2) Answer how many numbers between indices A and B (inclusive) are divisible by 3. This is represented by the command "1 A B".

这题是显然的维护分治状态的题，对于每一个区间维护除 3 余 0，1，2 的个数即可.

```
#include <iostream>
#include <string.h>
using namespace std;

const int maxn = 1e5 + 10;
struct Node{
  int cnt[3];
  int lazy;
} tree[maxn << 2];
      int N , Q;

      void build(int l , int r , int id){
        if(l > r)return;
        if(l == r){
          tree[id].cnt[0] = 1;
        }
        else{
          int m = (l + r) >> 1;
          build(l , m , id << 1);
          build(m + 1 , r , id << 1 | 1);
          for(int i = 0; i < 3; ++i){
            tree[id].cnt[i] = tree[id << 1].cnt[i] + tree[id << 1 | 1].cnt[i];
          }
        }
      }

      Node merge(const Node& a , const Node& b){
        Node ret;
        ret.lazy = 0;
        for(int i = 0; i < 3; ++i)
          ret.cnt[i] = a.cnt[i] + b.cnt[i];
        return ret;
      }

      void push_down(int l , int r , int id){
        if(l < r && tree[id].lazy){
          Node ret; // fix left child
          for(int i = 0; i < 3; ++i){
            ret.cnt[i] = tree[id << 1].cnt[(i + tree[id].lazy) % 3];
          }
          ret.lazy = tree[id << 1].lazy + tree[id].lazy;
          tree[id << 1] = ret;
          //fix right child
          for(int i = 0; i < 3; ++i){
            ret.cnt[i] = tree[id << 1 | 1].cnt[(i + tree[id].lazy) % 3];
          }
          ret.lazy = tree[id << 1 | 1].lazy + tree[id].lazy;
          tree[id << 1 | 1] = ret;
          tree[id].lazy = 0;
        }
      }

      void update(int l , int r , int x , int y , int id){
        if(l > r || l > y || r < x)return;
        if(l <= x && y <= r){
          Node ret;
          for(int i = 0; i < 3; ++i){
            ret.cnt[i] = tree[id].cnt[(i + 1) % 3];
          }
          ret.lazy = tree[id].lazy + 1;
          tree[id] = ret;
        }
        else{
          int m = (x + y) >> 1;
          push_down(x , y , id);
          update(l , r , x , m , id << 1);
          update(l , r , m + 1 , y , id << 1 | 1);
          tree[id] = merge(tree[id << 1] , tree[id << 1 | 1]);
        }
      }

      Node query(int l , int r , int x , int y , int id){
        if(l <= x && y <= r){
          return tree[id];
        }
        int m = (x + y) >> 1;
        push_down(x , y , id);
        if(m < l)
          return query(l , r , m + 1 , y , id << 1 | 1);
        if(m >= r)
          return query(l , r , x , m , id << 1);
        return merge(query(l , r , x , m , id << 1) , query(l , r , m + 1 , y , id << 1 | 1));
      }

      int main(){
        ios::sync_with_stdio(false);
        cin.tie(nullptr);
        cin >> N >> Q;
        memset(tree , 0 , sizeof(tree));
        build(0 , N - 1 , 1);
        for(int q = 1; q <= Q; ++q){
          int op; cin >> op;
          if(op == 0){
            int l , r; cin >> l >> r;
            update(l , r , 0 , N - 1 , 1);
          }
          if(op == 1){
            int l , r; cin >> l >> r;
            Node ret = query(l , r , 0 , N - 1 , 1);
            cout << ret.cnt[0] << endl;
          }
        }
      }
```

DP + segment tree optimization:
Codeforces-474 Pillar
Problem statement:
Marmot found a row with *n* pillars. The *i*-th pillar has the height of h_i meters. Starting from one pillar i_1, Marmot wants to jump on the pillars i_2, ..., i_k. ($1 \le i_1 < i_2 < ... < i_k \le n$). From a pillar *i* Marmot can jump on a pillar *j* only if $i<j$ and $|h_i-h_j| \ge d$, where $|x|$ is the absolute value of the number *x*.

Now Marmot is asking you find out a jump sequence with maximal length and print it.

Solution:
DP + segment tree
Sweep from left to right
Let dp[i] = dp[j] + 1, where j < i and |h[i]-h[j]|>=d
We have O(N^2) solution now.
Now we use segment tree to optimize it.
Discretise all h[i], so that we have at most 10^5 of them and build segment tree over them.
Tree[l , r] will keep track of the maximal DP value and index for l <= h <= r.
Each time we just query for max in [0 , h[i] - d] and [h[i] + d , N] and we are done.

Discretise: sort points , remove duplicate and keep a value -> id mapping.

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <unordered_map>
#include <cstdio>
using namespace std;
typedef pair<int , int> ii;
typedef long long ll;
```

```cpp
const int inf = 50000000;
const int maxn = 1e5 + 10;
ll H[maxn] , Hs[maxn] , Hss[maxn];
unordered_map<ll , int> mp;
int dp[maxn << 2][2];
int dp2[maxn];
int pre[maxn];
int N , D;

void update(int l , int r , int pos , int val , int index , int id){
  if(l > r || pos < l || pos > r)return;
  if(l == r && l == pos){
    dp[id][0] = val;
    dp[id][1] = index;
  }
  else{
    int m = (l + r) >> 1;
    update(l , m , pos , val , index , id << 1);
    update(m + 1 , r , pos , val , index , id << 1 | 1);
    dp[id][0] = (dp[id << 1][0] > dp[id << 1 | 1][0]) ? dp[id << 1][0] : dp[id << 1
| 1][0];
    dp[id][1] = (dp[id << 1][0] > dp[id << 1 | 1][0]) ? dp[id << 1][1] : dp[id << 1
| 1][1];
    //printf("(%d %d) best  is (%d %d)\n" , l , r , dp[id][0] , dp[id][1]);
  }
}

int query(int l , int r , int x , int y , int id){
  if(y < l || x > r){ // out of range
    return -1;
  }
  else if (x >= l && y <= r){ // in the range
    return dp[id][1];
  }
  int m = (x + y) >> 1;
  int left = query(l , r , x , m , id << 1);
  int right = query(l , r , m + 1 , y , id << 1 | 1);
  int ret = (dp2[left] > dp2[right]) ? left : right;
  return ret;
}

int main(){
  cin >> N >> D;
  for(int i = 0; i < N; ++i){
    cin >> H[i];
    Hss[i] = H[i]; // Hs will be sorted version of Hs
    pre[i] = -1; // ini pre
    dp2[i] = 0;
  }
  sort(Hss , Hss + N);
  int tot = 0; // total number of distinct number in H
  mp[Hss[0]] = tot;
  Hs[0] = Hss[0];
  for(int i = 1; i < N; ++i){
    if(Hss[i] != Hss[i - 1]){
      ++tot;
      mp[Hss[i]] = tot;
      Hs[tot] = Hss[i];
    }
  }
  for(int i = 0; i <= (N << 2); ++i){
    dp[i][0] = 0;
    dp[i][1] = -1;
  }
  for(int i = 0; i < N; ++i){
    auto j = lower_bound(Hs , Hs + tot , H[i] + D);
    auto k = lower_bound(Hs , Hs + tot , H[i] - D);
    if(*k > H[i] - D){
      if(k == Hs)k = nullptr;//left bound element is larger than target -> no such element in array
      else --k;
    }
    if(*j < H[i] + D)j = nullptr; //right bound element is smaller than target -> no such bound in array
    int left = -1 , right = -1;
    if(j != nullptr) right = query(mp[*j] , tot , 0 , tot , 1);
    if(k != nullptr) left = query(0 , mp[*k] , 0 , tot , 1);
    int best = -1;
    if(right == -1)best = left;
    if(left == -1)best = right;
    if(left > -1 && right > -1)best = (dp2[left] > dp2[right]) ? left : right;
    //cout << i << " " << best << "  " << left << "  " << right << endl;
    if(best >= 0){
      pre[i] = best;
      dp2[i] = 1 + dp2[best];
    }
    else dp2[i] = 1;
    //printf("dp2[%d] = %d\n" , i , dp2[i]);
    update(0 , tot , mp[H[i]] , dp2[best] + 1 , i , 1);
  }
  cout << dp[1][0] << endl;
  int start = dp[1][1];
  vector<int> path;
  while(start != -1){
    path.push_back(start);
    start = pre[start];
  }
  reverse(path.begin() , path.end());
  for(int i = 0; i < path.size(); ++i){
    cout << path[i] + 1 << " ";
  }
  cout << endl;
  return 0;
}
```

Sweeping

Sweeping is usually done intuitively.
When sweeping, think about the ordering of the things that you want to sweep, think about which direction we might sweep.(left to right / right to left / bottom to top / top to bottom?)

Sweeping can be very hard, but usually when ordering is not important, we might want to sort something, and sweeping might comes in naturally.
Examples:

Codeforces-524E – Rooks and rectangles

Given a N by N chess board, and position of k rooks, given Q query, each ask if the rectangle of top left = (x1 , y1) and right bottom = (x2 , y2) can be “fully covered by rooks only in this rectangle”.

Solution: when a rectangle is fully covered, either each of it’s rows or columns have at least one rook on it, so we can check rows and columns independently.
Say we are check rows now, firstly , sort the query by right point, so when we sweep from left to right (on x-axis), we make sure that all rooks are from the left of the right point(定右界). Let there be a segment tree that dictate the minimal x for a given range [yl , yr].
(扫 x 轴 在 y 轴建立线段树 维护最小 x)
When checking if all rows are covered, simply query(yl , yr), see if the return value is at least the x value of left-top point.

Similar for columns check.

Code:

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <cstdio>
using namespace std;
typedef pair<int , int> ii;

const int inf = 1 << 28;
const int maxn = 2e5 + 10;
int N , M , K , Q;
ii pos[maxn];
int tree[maxn << 2];
bool ans[maxn];
struct query{
  ii a , b;
  int id;
  bool operator<(const query& other) const{
    return b < other.b;
  }
}q[maxn];
/*
given a NxM chessboard and positions of K rooks.
given Q rectangles , we need to determine if in each of the rectangles that each
ceil can be attacked by some rook that are also in this rectangle.

*/

void update(int pos , int val , int l , int r , int id){
  if(l > r || pos > r || pos < l)return;
  if(l == r){
    tree[id] = max(tree[id] , val);
  }
  else{
    int m = (l + r) >> 1;
    update(pos , val , l , m , id << 1);
    update(pos , val , m + 1 , r , id << 1 | 1);
    tree[id] = min(tree[id << 1] , tree[id << 1 | 1]);
  }
}

int query(int l , int r , int x , int y , int id){
  if(l <= x && y <= r){
    return tree[id];
  }
  int m = (x + y) >> 1;
  if(m < l){
    return query(l , r , m + 1 , y , id << 1 | 1);
  }
  if(m >= r){
    return query(l , r , x , m , id << 1);
  }
  return min(query(l , r , x , m , id << 1) , query(l , r , m + 1 , y , id << 1 |
1));
}

void solve(){
  sort(pos + 1 , pos + 1 + K);
  sort(q + 1 , q + Q + 1);
  int j = 1;
  int up = max(N , M);
  //for(int i = 1; i <= 2 * up; ++i)tree[i] = 0;
  memset(tree , 0 , sizeof(tree));
  for(int i = 1; i <= Q; ++i){
    //printf("q : (%d %d)\n" , q[i].b.first , q[i].b.second);
    while(j <= K && pos[j] <= q[i].b)update(pos[j].second , pos[j].first , 1 , up ,
1) , ++j;
    int miny = query(q[i].a.second , q[i].b.second , 1 , up , 1);
    //printf("q(%d %d) = %d\n" , q[i].a.first , q[i].b.first , miny);
    if(miny >= q[i].a.first)ans[q[i].id] = true;
  }
}


int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> N >> M >> K >> Q;
  for(int i = 1; i <= K; ++i){
    cin >> pos[i].first >> pos[i].second;
  }
  for(int i = 1; i <= Q; ++i){
    cin >> q[i].a.first >> q[i].a.second;
    cin >> q[i].b.first >> q[i].b.second;
    q[i].id = i;
    ans[i] = false;
  }
  solve(); //check for columns first
  // swap rows and cols
  for(int i = 1; i <= K; ++i){
    swap(pos[i].first , pos[i].second);
  }
  for(int i = 1; i <= Q; ++i){
    swap(q[i].a.first , q[i].a.second);
    swap(q[i].b.first , q[i].b.second);
  }
  solve(); //check for rows
  for(int i = 1; i <= Q; ++i){
    if(ans[i])cout << "YES\n";
    else cout << "NO\n";
  }
}
```

2019 Google kick-start round B question C

有点像 last occurrence，因为是对于每一个元素最多出现 S 次，所以在 pre[S][A[i]] 的位置设置为-S，位置 i 设为+1，pre[S+1][A[i]]设置为 0，然后 query A[1..i]中的最大连续数组和即可

用到的知识点为：正负抵消，维护分治状态

```cpp
#include <iostream>
#include <string.h>
#include <algorithm>
#include <unordered_map>
using namespace std;
const int maxn = 1e5 + 10;
int N , S;
int A[maxn];
int pre[maxn] , ipre[maxn];
      int P[maxn];
      /*
        similar to last occurence , but last S occurence
        notice that it is impossible to generate solution by sliding window,
        so we need to consider all possible [l , r]
        but this is O(N ^ 2) , one trick is to fix r , and find optimal l
        each time we consider A[r] , set the sum at r be +1 , and pre[S][r] in sum be -S
        and pre[S + 1][r] be 0
        and we just query the maximal contiguous sum in A[1...r]
        This can be done by segment tree, each node maintains the largest sum starting from l,
        largest sum ending at r, actual sum of A[l...r], and finally, the solution in A[l...r]
        overrall time complexity = O(NlogN)
      */
      struct state{
        int L , R , sum , best;
      } tree[maxn << 2];

      state merge(const state& left , const state& right){
        state ret;
        ret.sum = left.sum + right.sum;
        ret.L = max(left.L , left.sum + right.L);
        ret.R = max(right.R , left.R + right.sum);
        ret.best = max(left.best , right.best);
        ret.best = max(ret.best , left.R + right.L);
        return ret;
      }

      void update(int pos , int l , int r , int id , int val){
        if(pos < l || pos > r)return;
        if(pos == l && r == pos){
          tree[id].sum = val;
          tree[id].L = tree[id].R = tree[id].best = max(0 , val);
        }
        else{
          int m = (l + r) >> 1;
          if(pos <= m)
            update(pos , l , m , id << 1 , val);
          else
            update(pos , m + 1 , r , id << 1 | 1 , val);
          tree[id] = merge(tree[id << 1] , tree[id << 1 | 1]);
        }
      }
```

```cpp

state query(int l , int r , int x , int y , int id){
  if(l <= x && y <= r){
    return tree[id];
  }
  int m = (x + y) >> 1;
  state ret;
  if(m >= r){
    ret = query(l , r , x , m , id << 1);
    return ret;
  }
  if(m < l){
    ret = query(l , r , m + 1 , y , id << 1 | 1);
    return ret;
  }
  ret = merge(query(l , r , x , m , id << 1) , query(l , r , m + 1 , y , id << 1 | 1));
  return ret;
}

void init(){
  memset(P , 0 , sizeof(P));
  memset(pre , 0 , sizeof(pre));
  memset(ipre , 0 , sizeof(ipre));
  unordered_map<int , int> mp , fc , cnt; // first occurence
  for(int i = 1; i <= N; ++i){
    if(!fc[A[i]])fc[A[i]] = i;
    if(mp[A[i]]){
      pre[i] = mp[A[i]];
      ipre[pre[i]] = i;
    }
    mp[A[i]] = i;
    ++cnt[A[i]];
    if(cnt[A[i]] > S){
      P[i] = fc[A[i]];
      fc[A[i]] = ipre[fc[A[i]]];
    }
  }
}

int solve(){
  int ret = 0;
  memset(tree , 0 , sizeof(0));
  for(int i = 1; i <= N; ++i){
    update(i , 1 , N , 1 , 1);
    update(P[i] , 1 , N , 1 , -S);
    update(pre[P[i]] , 1 , N , 1 , 0);
    state rr = query(1 , i , 1 , N , 1);
    ret = max(ret , rr.best);
  }
  return ret;
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int T;
  cin >> T;
  for(int t = 1; t <= T; ++t){
    cin >> N >> S;
    for(int i = 1; i <= N; ++i){
            cin >> A[i];
        }
        init();
        cout << "Case #" << t << ": " << solve() << endl;
    }
    return 0;
}
```

More on segment trees:

1.build segment tree over trees.
Build segment tree corresponding to each node's DFS order. So, each subtree is just an interval.

2.Range updates & range query
Lazy tags on each node, the invariant is, when we terminate on a certain node in range tree, We know that we are "safe" at this point, being this node will store the correct answer for this range. When we go down in tree while updating or querying, always remember to push down the lazy tag to child and pull up when coming back.
(push down lazy tag , change child's value according to parent's tag , set parent's tag to untag finally)

Here is an example where we use both of these.

Codeforces-396C On changing tree

You are given a rooted tree consisting of n vertices numbered from 1 to n. The root of the tree is a vertex number 1.

Initially all vertices contain number 0. Then come q queries, each query has one of the two types:

- The format of the query: $1\ v\ x\ k$. In response to the query, you need to add to the number at vertex v number x; to the numbers at the **descendants** of vertex v at distance 1, add x - k; and so on, to the numbers written in the descendants of vertex v at distance i, you need to add x - $(i{\cdot}k)$. The distance between two vertices is the number of edges in the shortest path between these vertices.
- The format of the query: $2\ v$. In reply to the query you should print the number written in vertex v modulo 1000000007 $(10^9 + 7)$.

Solution:

when adding x on vertex v, it's descendant c will have value $x - k * (level[c] - level[v]) = (x + k * level[v]) - k * level[c]$. Note that the things we need to store for c is just it's level,
$x + k * level[v]$ and $k.level[c]$ is easy, $x + k * level[v]$ and k can be stored in 2 segment trees using lazy propagation over the interval of the subtree of v.

Code:

```
#include <iostream>
#include <vector>
#include <string.h>
using namespace std;

typedef long long ll;
const ll mod = 1e9 + 7;
const int maxn = 3e5 + 10;
int N , Q;
vector<int> g[maxn];
```

```cpp
struct Node{
  ll sum , lazy;
} tree[maxn << 2][2]; //val0 = x + k * level[parent] , val1 = k
int level[maxn];
int dfn[maxn];
int L[maxn];
int tot;

void DFS(int v , int p , int d){
  level[v] = d;
  dfn[v] = tot++;
  for(int u : g[v])
    if(u != p)
      DFS(u , v , d + 1);
  L[dfn[v]] = tot - 1;
}

void push_down(int l , int r , int id , int ver){
  if(l < r && tree[id][ver].lazy){
    tree[id << 1][ver].lazy = (tree[id << 1][ver].lazy + tree[id][ver].lazy) % mod;
    tree[id << 1 | 1][ver].lazy = (tree[id << 1 | 1][ver].lazy + tree[id][ver].lazy) % mod;
    tree[id << 1][ver].sum = (tree[id << 1][ver].sum + tree[id][ver].lazy) % mod;
    tree[id << 1 | 1][ver].sum = (tree[id << 1 | 1][ver].sum + tree[id][ver].lazy) % mod;
    tree[id][ver].lazy = 0;
  }
}

void merge(int l , int r , int id , int ver){
  if(l < r){
    tree[id][ver].sum = (tree[id << 1][ver].sum + tree[id << 1 | 1][ver].sum) % mod;
  }
}

void update(int l , int r , int x , int y , int id , ll val , int ver){
  if(l > r || l > y || r < x)return;
  if(l <= x && y <= r){
    tree[id][ver].sum = (tree[id][ver].sum + val) % mod;
    tree[id][ver].lazy = (tree[id][ver].lazy + val) % mod;
  }
  else{
    int m = (x + y) >> 1;
    push_down(x , y , id , ver);
    update(l , r , x , m , id << 1 , val , ver);
    update(l , r , m + 1 , y , id << 1 | 1 , val , ver);
    merge(x , y , id , ver);
  }
}

ll query(int l , int r , int x , int y , int id , int ver){
  if(l <= x && y <= r){
    return tree[id][ver].sum;
  }
  int m = (x + y) >> 1;
  push_down(x , y , id , ver);
  if(m < l)
    return query(l , r , m + 1 , y , id << 1 | 1 , ver);
  if(m >= r)
    return query(l , r , x , m , id << 1 , ver);
  ll ret = 0;
  ret = (ret + query(l , r , x , m , id << 1 , ver)) % mod;
  ret = (ret + query(l , r , m + 1 , y , id << 1 | 1 , ver)) % mod;
  return ret;
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> N;
  for(int i = 2; i <= N; ++i){
    int p; cin >> p;
    g[p].push_back(i);
    g[i].push_back(p);
  }
  tot = 1;
  DFS(1 , 1 , 0);
  cin >> Q;
  memset(tree , 0 , sizeof(tree));
  for(int q = 1; q <= Q; ++q){
    int tp; cin >> tp;
    if(tp == 1){
      ll v , x , k; cin >> v >> x >> k;
      ll val = k * level[v] % mod;
      val = (val + x) % mod;
      update(dfn[v] , L[dfn[v]] , 1 , N , 1 , val , 0);
      update(dfn[v] , L[dfn[v]] , 1 , N , 1 , k , 1);
    }
    if(tp == 2){
      int v; cin >> v;
      ll a1 = query(dfn[v] , dfn[v] , 1 , N , 1 , 0);
            ll a2 = query(dfn[v] , dfn[v] , 1 , N , 1 , 1) * level[v] % mod;
            cout << (a1 - a2 + mod) % mod << endl;
          }
        }
      }
```

3.Sometime, we can build segment tree over queries!!!!
(When some operation has 时效性, one can interpret it as : this operation is available during [tl , tr])

Example problem:
1.Assn1 part2 (b), offline dynamic connectivity.
对于 query 建树，每一条边有实效性，DFS 到线段树的每一个 leaf，向下的过程中用 union-find + stack 构造联通关系，回溯时利用(a)的“删除最近加入的边”的方法删除在这个时段内加入的所有边。这相当于在“时间”上图的维护。
此题目也用到了 union by size，将小的连到大的。这样可以保证高度为 logN。
因为从 child 到祖先的路上，每向上一次，新的根掌管的子树大小至少翻两倍

Code:
```
#include <iostream>
#include <vector>
using namespace std;
typedef pair<int , int> ii;

const int maxn = 1e5 + 10;
int N , Q;
struct query{
  int c; //0->add , 1->query
  int a , b; //node a and b
  int start , end; //only for add
  query(){}
  query(int c_ , int a_ , int b_ , int s_ , int e_) : c(c_) , a(a_) , b(b_) , start(s_) , end(e_) {}
}qs[maxn];
vector<ii> tree[maxn << 2];
// for each segment tree node , store the edges that are legal for this range
int ans[maxn];
// store answer for each query
vector<int> a[maxn];
vector<int> csize[maxn];
vector<ii> q;

//from part 2 a

int find(int x){
  return a[x].back() == x ? x : find(a[x].back());
}

void undo(){
  int x = q.back().first;
  int y = q.back().second;
  q.pop_back();
  a[x].pop_back();
  a[y].pop_back();
  csize[x].pop_back();
  csize[y].pop_back();
}

void add(int x , int y){
  x = find(x);
  y = find(y);
  if(x != y){ // different ancestor
    if(csize[x].back() > csize[y].back())swap(x , y); //link the smaller one to larger one
    a[x].push_back(y);
    a[y].push_back(y);
    int sx = csize[x].back() , sy = csize[y].back();
    csize[x].push_back(sx + sy);
    csize[y].push_back(sx + sy);
  }
  else {
    a[x].push_back(x);
    a[y].push_back(y);
    csize[x].push_back(csize[x].back());
    csize[y].push_back(csize[y].back());
  }
  q.emplace_back(x , y);
}

void DFS(int id , int l , int r){
  if(l > r)return;
  int cnt = 0;
  for(auto& it : tree[id]){
    add(it.first , it.second);
    ++cnt;
  }
  if(l == r){
    if(qs[l].c)
      ans[l] = find(qs[l].a) == find(qs[l].b);
  }
  else{
    int mid = (l + r) >> 1;
    DFS(id << 1 , l , mid);
    DFS(id << 1 | 1 , mid + 1 , r);
  }
  while(cnt){
    undo();
    --cnt;
  }
}

void update(int l , int r , int x , int y , int a , int b , int id){
  if(l > r || x > y || l > y || r < x)return;
  if(l <= x && y <= r){
    tree[id].emplace_back(a , b);
  }
  else{
    int mid = (x + y) >> 1;
    update(l , r , x , mid , a , b , id << 1);
    update(l , r , mid + 1 , y , a , b , id << 1 | 1);
  }
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> N >> Q;
  for(int i = 1; i <= Q; ++i){
    char op;
    cin >> op;
    if(op == 'A'){
      int x , y;
      cin >> x >> y;
      qs[i] = query(0 , x , y , i , Q);
    }
    else if(op == 'D'){
      int x;
      cin >> x;
      qs[x].end = i;
    }
    else{
      int x , y;
      cin >> x >> y;
      qs[i] = query(1 , x , y , 0 , 0);
    }
    ans[i] = -1;
  }
  for(int i = 1; i <= Q * 2; ++i)tree[i].clear();
  for(int i = 1; i <= N; ++i){
    a[i].clear();
    a[i].push_back(i);
    csize[i].clear();
    csize[i].push_back(1);
  }
  for(int i = 1; i <= Q; ++i){
    if(qs[i].c == 0){
      //add into segment tree node
      update(qs[i].start , qs[i].end , 1 , Q , qs[i].a , qs[i].b , 1);
    }
  }
  q.clear();
  DFS(1 , 1 , Q);
  for(int i = 1; i <= Q; ++i){
    if(ans[i] != -1)
      cout << ans[i] << endl;
  }
}
```

2.**Codeforces-258E Little elephant and tree**

Problem statement:
Given a tree with N nodes, initially, each node is empty. Follow by M operations, each has the form (a , b) , meaning, put number i in all nodes in subtrees rooted at ai or bi. Finally, For each node v , count cv, where cv is the number of nodes that share at least 1 number with v.
Solution:
For reach node, store the operations at this node, cover all their subtree with +1 tag, query for the whole tree then DFS down. When come back, cover all their subtree with -1 tag.
线段覆盖！

Code:

```cpp
#include <iostream>
#include <vector>
#include <string.h>
using namespace std;

const int maxn = 1e5 + 10;
vector<int> g[maxn];
vector<int> qs[maxn];
int N , M;
int ans[maxn];
int dfn[maxn];
int R[maxn];
int tot;
struct Node{
  int sum , lazy;
} tree[maxn << 2];


void DFS1(int v , int p){
  dfn[v] = tot++;
  for(int u : g[v])
    if(u != p)
      DFS1(u , v);
  R[dfn[v]] = tot - 1;
}

void update(int l , int r , int x , int y , int id , int val){
  if(x > y || l > r || l > y || r < x)return;
  if(l <= x && y <= r){
    tree[id].lazy += val;
    if(tree[id].lazy){
      tree[id].sum = y - x + 1;
    }
    else{
      if(x == y)tree[id].sum = 0;
      else tree[id].sum = tree[id << 1].sum + tree[id << 1 | 1].sum;
    }
  }
  else{
    int m = (x + y) >> 1;
    update(l , r , x , m , id << 1 , val);
    update(l , r , m + 1 , y , id << 1 | 1 , val);
    if(tree[id].lazy){
      tree[id].sum = y - x + 1;
    }
    else{
      if(x == y)tree[id].sum = 0;
      else tree[id].sum = tree[id << 1].sum + tree[id << 1 | 1].sum;
    }
  }
}

void DFS2(int v , int p){
  for(int u : qs[v]){
    update(dfn[u] , R[dfn[u]] , 1 , N , 1 , 1); // cover
  }
  ans[v] = tree[1].sum;
  for(int u : g[v])
    if(u != p)
      DFS2(u , v);
  for(int u : qs[v]){
    update(dfn[u] , R[dfn[u]] , 1 , N , 1 , -1);
  }
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> N >> M;
  for(int i = 1; i <= N - 1; ++i){
    int x , y; cin >> x >> y;
    g[x].push_back(y);
    g[y].push_back(x);
  }
  for(int i = 1; i <= M; ++i){
    int x , y; cin >> x >> y;
    qs[x].push_back(y);
    qs[x].push_back(x);
    qs[y].push_back(x);
    qs[y].push_back(y);
  }
  tot = 1;
  memset(tree , 0 , sizeof(tree));
  DFS1(1 , 1);
  DFS2(1 , 1);
  for(int i = 1; i <= N; ++i){
    if(ans[i])cout << ans[i] - 1 << " ";
    else cout << ans[i] << " ";
  }
  cout << endl;
}
```

4. Range tree of data structures.
The use of order statistic tree.
Example problem:
SPOJ-ADABERRY

Given a tree, each node will store some numbers, query is to answer for some node v and x, how many numbers are less than x in the subtree rooted at v.
Solution:
Range tree + order statistic tree
Each Node in range tree will store the numbers that appear in it’s subtree.
Query complexity = update complexity = O(N * logN * logN)
Code:

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds ;
using namespace std ;
typedef tree <int , null_type , less <int >, rb_tree_tag ,
tree_order_statistics_node_update> ordered_set ;

const int maxn = 2e5 + 10;
    int N , Q;
    ordered_set s[maxn << 2]; //ordered statistic tree used as segment tree on tree
    int dfn[maxn]; //dfs ordering
    int R[maxn]; //the interval
    int a[maxn]; //old value on each node
    int mp[maxn]; //inverse of dfn for each v
    bool vis[maxn];
    vector<int> g[maxn];
    int cnt;

    void DFS(int v){
        vis[v] = true;
        dfn[v] = cnt;
        mp[cnt] = v;
        ++cnt;
        for(int i : g[v]){
            if(!vis[i])
                DFS(i);
        }
        R[dfn[v]] = cnt - 1;
    }


    void build(int l , int r , int id){
        if(l > r)return;
        if(l == r){
            s[id].insert(a[mp[l]]);
        }
        else{
            int m = (l + r) >> 1;
            build(l , m , id << 1);
            build(m + 1 , r , id << 1 | 1);
            for(int i : s[id << 1]){
                s[id].insert(i);
            }
            for(int i : s[id << 1 | 1]){
                s[id].insert(i);
            }
        }
    }

    void update(int l , int r , int x , int y , int val , int id){
        if(l > r || y < l || x > r)return;
        if(l <= x && y <= r){
            s[id].insert(val);
        }
        else{
            int m = (x + y) >> 1;
            update(l , r , x , m , val , id << 1);
            update(l , r , m + 1 , y , val , id << 1 | 1);
            s[id].insert(val);
        }
    }

    int query(int l , int r , int x , int y , int val , int id){
        if(l > r || y < l || x > r)return 0;
        if(l <= x && y <= r){
            return s[id].order_of_key(val);
        }
        else{
            int m = (x + y) >> 1;
            int ret = 0;
            ret += query(l , r , x , m , val , id << 1);
            ret += query(l , r , m + 1 , y , val , id << 1 | 1);
            return ret;
        }
    }

    int main(){
        scanf("%d %d", &N , &Q);
        for(int i = 0; i < N; ++i){
            scanf("%d", &a[i]);
            g[i].clear();
            vis[i] = false;
        }
        for(int i = 0; i < N - 1; ++i){
            int x , y;
            scanf("%d %d", &x , &y);
            g[x].push_back(y);
            g[y].push_back(x);
        }
        cnt = 0;
        DFS(0);
        build(0 , N - 1 , 1);
        for(int q = 1; q <= Q; ++q){
            int v , val;
            scanf("%d %d" , &v , &val);
            printf("%d\n" , query(dfn[v] , R[dfn[v]] , 0 , N - 1 , val , 1));
            update(dfn[v] , dfn[v] , 0 , N - 1 , val , 1);
        }
        return 0;
    }
```

5. **Searching in a range tree.**
Example problem:
Codeforces-19D-Points
Problem statement:

Pete and Bob invented a new interesting game. Bob takes a sheet of paper and locates a Cartesian coordinate system on it as follows: point $(0, 0)$ is located in the bottom-left corner, Ox axis is directed right, Oy axis is directed up. Pete gives Bob requests of three types:

- `add x y` — on the sheet of paper Bob marks a point with coordinates (x, y). For each request of this type it's guaranteed that point (x, y) is not yet marked on Bob's sheet at the time of the request.
- `remove x y` — on the sheet of paper Bob erases the previously marked point with coordinates (x, y). For each request of this type it's guaranteed that point (x, y) is already marked on Bob's sheet at the time of the request.
- `find x y` — on the sheet of paper Bob finds all the marked points, lying strictly above and strictly to the right of point (x, y). Among these points Bob chooses the leftmost one, if it is not unique, he chooses the bottommost one, and gives its coordinates to Pete.

Solution:
build range tree over the x-axis, for each [xl , xr], maintain the maximal y.
Also maintain a set, X[i] will store all the ys at x = i.
When query for (x , y) , query(x + 1 , N) , search in range tree, find the smallest xi such that the maximal it has is greater than y. Say we have found xi, to find yi, we just need to find the smallest element greater than y in X[xi].
Code:

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
#include <vector>
#include <set>
#include <algorithm>
#include <unordered_map>
using namespace std;

const int maxn = 2e5 + 10;
const int inf = 1 << 28;
int Q[maxn][3];
int N;
int cnt;
int tree[maxn << 2];
unordered_map<int , int> mp, imp;
set<int> Y[maxn];

/*
  segment tree maintains the maximal y for a range
  Y[x] has all the ys on this x
*/

void build(int l , int r , int id){
  if(l > r)return ;
  if(l == r){
    if(Y[l].size() == 0)tree[id] = -1;
    else tree[id] = *Y[l].rbegin();
  }
  else{
    int m = (l + r) >> 1;
    build(l , m , id << 1);
    build(m + 1 , r , id << 1 | 1);
    tree[id] = max(tree[id << 1] , tree[id << 1 | 1]);
  }
}

void add(int pos , int val , int l , int r , int id){
  if(l > r)return;
  if(pos == l && l == r){
    tree[id] = max(tree[id] , val);
  }
  else{
    int m = (l + r) >> 1;
    if(pos <= m)add(pos , val , l , m , id << 1);
    else add(pos , val , m + 1 , r , id << 1 | 1);
    tree[id] = max(tree[id << 1] , tree[id << 1 | 1]);
  }
}

int query(int l , int r , int x , int y , int val , int id){
  if(l > r || x > r || y < l)return inf;
  if(tree[id] <= val)return inf;
  if(l <= x && y <= r){
    if(x == y)return x;
    else{
      int m = (x + y) >> 1;
      if(tree[id << 1] > val)return query(l , r , x , m , val , id << 1);
      return query(l , r , m + 1 , y , val , id << 1 | 1);
    }
  }
  else{
    int m = (x + y) >> 1;
    return min(query(l , r , x , m , val , id << 1) , query(l , r , m + 1 , y ,
val , id << 1 | 1));
  }
}
```

```cpp

void fix(int pos , int x , int y , int id){
  if(x > y)return;
  if(pos == x && x == y){
    if(Y[x].size() > 0)tree[id] = *(Y[x].rbegin());
    else tree[id] = -1;
  }
  else{
    int m = (x + y) >> 1;
    if(pos <= m)fix(pos , x , m , id << 1);
    else fix(pos , m + 1 , y , id << 1 | 1);
    tree[id] = max(tree[id << 1] , tree[id << 1 | 1]);
  }
}

int main(){
  scanf("%d", &N);
  vector<int> X;
  for(int i = 1; i <= N; ++i){
    string t;
    cin >> t;
    scanf("%d %d" , &Q[i][0] , &Q[i][1]);
    X.push_back(Q[i][0]);
    if(t == "add"){
      Q[i][2] = 1;
    }
    if(t == "find"){
      Q[i][2] = 2;
    }
    if(t == "remove"){
      Q[i][2] = 3;
    }
  }
  sort(X.begin() , X.end());
  cnt = 1;
  mp[X[0]] = cnt;
  imp[cnt] = X[0];
  for(int i = 1; i < X.size(); ++i){
    if(X[i] != X[i - 1]){
      ++cnt;
      mp[X[i]] = cnt;
      imp[cnt] = X[i];
    }
  }
  build(1 , cnt , 1);
  for(int i = 1; i <= N; ++i){
    int x = mp[Q[i][0]] , y = Q[i][1];
    if(Q[i][2] == 1){ // add
      Y[x].insert(y);
      add(x , y , 1 , cnt , 1);
    }
    if(Q[i][2] == 2){ // find
      int ret = query(x + 1 , cnt , 1 , cnt , y , 1);
      if(ret != inf){
        if(Y[ret].upper_bound(y) != Y[ret].end())
          printf("%d %d\n" , imp[ret] , *Y[ret].upper_bound(y));
        else
          cout << -1 << endl;
      }
      else cout << -1 << endl;
    }
    if(Q[i][2] == 3) { // erase
      Y[x].erase(y);
      fix(x , 1 , cnt , 1);
    }
  }
  return 0;
}
```

Graphs

1.Graph traversal: BFS, DFS both O(|E| + |V|) -> trivial
2.**Bridge finding: DFS order & back edge.**
Example problem: Get all bridges in an undirected graph.
Code:

```
void DFS(int v , int p){
  dfn[v] = low[v] = tot++;
  for(int u : g[v]){
    if(u != p){
      if(dfn[u] != -1)low[v] = min(low[v] , dfn[u]);
      else{
        DFS(u , v);
        low[v] = min(low[v] , low[u]);
        if(dfn[u] == low[u])
          bridges.emplace_back(min(u , v) , max(u , v));
      }
    }
  }
}
```

3.**Find SCC in directed graph.**
Tarjan algorithm, Code:

```
void tarjan(int v){ //find strongly connected components in graph
  ++cnt;
  dfn[v] = low[v] = cnt; //increase dfn & low count
  s.push_back(v); //push to stack
  ins[v] = true; //in stack
  for(int i : g[v]){
    if(!dfn[i]){ //have not visited
      tarjan(i);
      low[v] = min(low[v] , low[i]);
    }
    else if(ins[i]){ //inside our stack
      low[v] = min(low[v] , dfn[i]);
    }
  }
  if(dfn[v] == low[v]){ // v is the root for this connected component !
    cout << "component : ";
    while(!s.empty()){ //everything above v in stack belong to the connected component
rooted at v
      cout << s.back() << " ";
      int k = s.back();
      s.pop_back();
      ins[k] = false;
      if(k == v)break;
    }
    cout << endl;
  }
}
```

4.**Cycle detection**
For a directed graph, if there is a back edge that passes my edge, then there must be a cycle. One can use the following method to check, we can also use tarjan to find SCC, and check if exists at least one SCC who's size is at least 2.

```
bool cycle(int v){
        if(vis[v])return false;
        vis[v] = active[v] = true;
        for(int u : g[v]){
                if(active[u] || cycle(u))
                        return true;
        }
        active[v] = false;
        return false;
}
```

5.DAG (directed acyclic graph)
If no cycle, then there exist a Maximal vertex and topological ordering.
Topsort: DFS, post order push vertex into vector, reverse when finish.
Reduce general graphs to DAG -> Tarjan.
Example problem:
Can of worms.
Code:

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <queue>
#include <vector>
#include <set>
#include <unordered_set>
using namespace std;
typedef long long ll;
/*
1. if Bi can reach Bj then Bi can reach all Bs between Bi and Bj
2. if Bi and Bj can reach each other then they can be viewed as a signle bomb

keys :
  1. generate a directed graph that has as least edges as possible
  2. run tarjan to find all SCC
  3. treat each SCC as a single point then dp
*/
```

```
const int maxn = 1e5 + 10;
int N;
ll X[maxn] , R[maxn];
vector<int> g[maxn]; //directed graph
unordered_set<int> G[maxn]; //SCC graph
int dfn[maxn] , low[maxn] , cnt , tot;
bool instack[maxn] , vis[maxn];
vector<int> S;
int component[maxn];
ll dp[maxn][2]; // left most and right most
int ans[maxn];

void DFS(int v){
  vis[v] = true;
  for(int u : G[v]){
    if(!vis[u])
      DFS(u);
    dp[v][0] = min(dp[v][0] , dp[u][0]);
    dp[v][1] = max(dp[v][1] , dp[u][1]);
  }
}

void tarjan(int v){
  dfn[v] = low[v] = cnt;
  S.push_back(v);
  instack[v] = true;
  ++cnt;
  for(int u : g[v]){
    if(dfn[u] == -1){
      tarjan(u);
      low[v] = min(low[v] , low[u]);
    }
    if(instack[u]){
      low[v] = min(low[v] , dfn[u]);
    }
  }
  if(low[v] == dfn[v]){
    while(!S.empty()){
      int b = S.back();
      component[b] = tot;
      dp[tot][0] = min(dp[tot][0] , X[b] - R[b]);
      dp[tot][1] = max(dp[tot][1] , X[b] + R[b]);
      S.pop_back();
      instack[b] = false;
      if(b == v)break;
    }
    ++tot;
  }
}

int main(){
  scanf("%d" , &N);
  vector<int> id;
  for(int i = 0; i < N; ++i){
    scanf("%lld %lld" , &X[i] , &R[i]);
    id.push_back(i);
    g[i].clear();
    G[i].clear();
    ans[i] = 0;
  }
  sort(id.begin() , id.end() , [&](int x , int y){
    return X[x] > X[y];
  }); //decreasing

  vector<ll> stack;
  for(int i = 0; i < N; ++i){
    int x = id[i];
    while(!stack.empty() && X[x] + R[x] >= X[stack.back()]){ //construct right edges
      g[x].push_back(stack.back());
      stack.pop_back();
    }
    stack.push_back(x);
  }

  sort(id.begin() , id.end() , [&](int x , int y){
    return X[x] < X[y];
  }); //increasing

  stack.clear();
  for(int i = 0; i < N; ++i){ //construct left edges
    int x = id[i];
    while(!stack.empty() && X[x] - R[x] <= X[stack.back()]){
      g[x].push_back(stack.back());
      stack.pop_back();
    }
    stack.push_back(x);
  }

  cnt = 0;
  tot = 0;
  for(int i = 0; i < N; ++i){
    dfn[i] = -1;
    low[i] = 1 << 28;
  instack[i] = false;
  dp[i][0] = 1ll << 40;
  dp[i][1] = -(1ll << 40);
 }
 for(int i = 0; i < N; ++i){
  if(dfn[i] == -1)
    tarjan(i);
 }
 // collapase each SCC into a point
 stack.clear();
 for(int i = 0; i < N; ++i){
  stack.push_back(X[i]);
  int c = component[i];
  for(int u : g[i]){
    int cc = component[u];
    if(c != cc)G[c].insert(cc);
  }
 }
 for(int i = 0; i < tot; ++i){
  vis[i] = false;
 }
 for(int i = 0; i < tot; ++i){
  if(!vis[i])
    DFS(i);
 }
 sort(stack.begin() , stack.end());
 for(int i = 0; i < tot; ++i){
  auto l = lower_bound(stack.begin() , stack.end() , dp[i][0]);
  auto h = upper_bound(stack.begin() , stack.end() , dp[i][1]);
  --h;
  ans[i] = h - l + 1;
 }
 for(int i = 0; i < N; ++i){
  cout << ans[component[i]] << " ";
 }
 cout << endl;
 return 0;
}
```

6. Minimum spanning tree.
How to find it? Kruskal's algorithm
Sort edges by their weight in increasing order, add edge into spanning forest if it does not create cycle (check by union find).

Example problem:
2019 IOI Day1 Q3 – Werefolf

Solution:
这题用到了很多知识，首先，建立两颗树，满足原来图的联通性质，且满足节点大小随深度增加而增加/减少的性质。比如说，建 parent 大于 child 的树时，永远保证 child 在 union-find 中的 parent 是当前的最大。建立线段树。
然后利用倍增跳到适合的位置，找到对应的线段。最后用类似 rooks and rectangles 的方法，将每一个点映射至坐标系中，query 等同于是给定一个矩形问能否包含至少一个点。

KEYS: minimum/maximum heap based on MST, binary flip, sweeping + range tree

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
typedef pair<int , int> ii;

const int maxn = 2e5 + 10;
vector<int> g[maxn];
vector<int> hightree[maxn]; // v greater than all child
vector<int> lowtree[maxn]; // v less than all child
int N , M , Q;
int ans[maxn];
int tree[maxn << 2];

struct unionset{
  int parent[maxn];
  void init(){
    for(int i = 1; i <= N; ++i)parent[i] = i;
  }

  int find(int x){
    return parent[x] == x ? x : parent[x] = find(parent[x]);
  }
}highH , lowH;

struct sTree{
  vector<int> t[maxn];
  int dfn[maxn];
  int tot;
  int right[maxn];
  int idfn[maxn];
  int up[maxn][20];
  int L;
  int jump(int start , int val , int high){
    int ret = start;
    if(high){
      for(int i = L; i >= 0; --i){
        if(up[ret][i] <= val)
          ret = up[ret][i];
      }
    }
    else{
      for(int i = L; i >= 0; --i){
        if(up[ret][i] >= val)
          ret = up[ret][i];
      }
    }
    return ret;
  }

  void DFS2(int v , int p){
    up[v][0] = p;
    for(int i = 1; i <= L; ++i)
      up[v][i] = up[up[v][i - 1]][i - 1];
    for(int u : t[v]){
      if(u != p)
        DFS2(u , v);
    }
  }

  void DFS(int v){ //DFS for dfs order , to turn tree into interval
    dfn[v] = ++tot;
    idfn[dfn[v]] = v;
    for(int u : t[v]){
      if(dfn[u] == -1){
        DFS(u);
      }
    }
    right[dfn[v]] = tot;
  }

  void process(int high){
    for(int i = 1; i <= N; ++i)dfn[i] = -1 , right[i] = -1 , idfn[i] = -1;
    tot = 0;
    // turn tree into interval by dfs order
    if(high)DFS(N);
    else DFS(1);
    L = ceil(log2(N));
    for(int i = 1; i <= N; ++i)
      for(int j = 0; j <= L; ++j)
```

```
        up[i][j] = 0;
    // calculate up[i][j]
    if(high)DFS2(N , N);
    else DFS2(1 , 1);
  }

  void init(){
    for(int i = 1; i <= N; ++i)t[i].clear();
  }

  void add(int x , int y){
    t[x].push_back(y);
  }

  void show(){
    for(int i = 1; i <= N; ++i){
      cout << i << " : ";
      for(int j : t[i]){
        cout << j << " ";
      }
      cout << endl;
    }
  }

}highT , lowT;

struct Query{
  int xl , xr , yl , yr , id;
  Query(){}
  Query(int a_ , int b_ , int c_ , int d_ , int e_): xl(a_) , xr(b_) , yl(c_) , yr(d_) , id(e_) {}
  bool operator < (const Query& other) const {
    return xr < other.xr;
  }
};

void build(int l , int r , int id){
  if(l > r)return;
  if(l == r){
    tree[id] = 0;
  }
  else{
    int m = (l + r) >> 1;
    build(l , m , id << 1);
    build(m + 1 , r , id << 1 | 1);
    tree[id] = max(tree[id << 1] , tree[id << 1 | 1]);
  }
}
```

```
void update(int pos , int x , int y , int val , int id){
  if(pos < x || pos > y)return;
  if(x == pos && y == pos){
    tree[id] = max(tree[id] , val);
  }
  else{
    int m = (x + y) >> 1;
    update(pos , x , m , val , id << 1);
    update(pos , m + 1 , y , val , id << 1 | 1);
    tree[id] = max(tree[id << 1] , tree[id << 1 | 1]);
  }
}

int query(int l , int r , int x , int y , int id){
  if(l <= x && y <= r){
    return tree[id];
  }
  int m = (x + y) >> 1;
  if(m < l)
    return query(l , r , m + 1 , y , id << 1 | 1);
  if(m >= r)
    return query(l , r , x , m , id << 1);
  return max(query(l , r , x , m , id << 1) , query(l , r , m + 1 , y , id << 1 | 1));
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> N >> M >> Q;
  for(int i = 1; i <= N; ++i)g[i].clear() , hightree[i].clear() , lowtree[i].clear();
  for(int i = 1; i <= M; ++i){
    int x , y;
    cin >> x >> y;
    ++x , ++y;
    g[x].push_back(y);
    g[y].push_back(x);
  }
  highH.init() , lowH.init();
  highT.init() , lowT.init();
  // build heap structure , where v is greater than all its child
  for(int i = 1; i <= N; ++i){ //for each connected component , parent is the maximal
    for(int j : g[i]){
      if(i > j){
        int y = highH.find(j); // y is maximal node in j's component
        if(i != y)highH.parent[y] = i , highT.add(i , y);
      }
    }
  }
  // build heap structure , where v is smaller than all its child
  for(int i = N; i >= 1; --i){ // for each connected component , parent is the minimal
    for(int j : g[i]){
      if(i < j){
        int y = lowH.find(j); // y is minimal node in j's component
        if(i != y)lowH.parent[y] = i , lowT.add(i , y);
      }
    }
  }
  highT.process(1);
  lowT.process(0);
  vector<ii> pos;
  for(int i = 1; i <= N; ++i){
    pos.emplace_back(highT.dfn[i] , lowT.dfn[i]); // (x , y)
  }
  vector<Query> qs;
  for(int i = 1; i <= Q; ++i){
    // for each S E L R , we can use binary flip to get subtree in logn
    // then the problem is turned into , given 2 permutations of 1 ... N
    // for [l1 , r1] , [l2 , r2] , if they have common element ?
    int S , E , L , R;
    cin >> S >> E >> L >> R;
    ++S , ++E , ++L , ++R;
    int v = highT.jump(E , R , 1); // jump from E , towards R
    int u = lowT.jump(S , L , 0); // jump from S , towards L
    int L1 = highT.dfn[v] , R1 = highT.right[highT.dfn[v]] , L2 = lowT.dfn[u] , R2 =
lowT.right[lowT.dfn[u]];
    qs.emplace_back(L1 , R1 , L2 , R2 , i);
    //(xl , xr , yl , yr)
  }
  sort(pos.begin() , pos.end());
  sort(qs.begin() , qs.end());
  build(1 , N , 1);
  int j = 0;
  for(int i = 0; i < Q; ++i){
    while(j < N && pos[j].first <= qs[i].xr){
      update(pos[j].second , 1 , N , pos[j].first , 1);
      ++j;
    }
    int ret = query(qs[i].yl , qs[i].yr , 1 , N , 1);
    ans[qs[i].id] = ret >= qs[i].xl;
  }
  for(int i = 1; i <= Q; ++i){
    cout << ans[i] << endl;
  }
```

```
}
```

7. Dijkstra’s algorithm
 Time Complexity O(E * logV)
Compute shortest distance between x and y, note that this only works for graph that has no negative edge.
How to choose next node to process? We choose the one among the unprocessed nodes, with the minimal distance.
We can do this using a priority queue.
Code:

```
void dijstra(int src){
  for(int i = 1; i < cnt; ++i){
    vis[i] = false;
    dis[i] = inf;
  }
  dis[src] = 0;
  priority_queue<pair<ll , int> , vector<pair<ll , int>> , greater<pair<ll , int>>> pq;
  pq.push(make_pair(dis[src] , src));
  while(!pq.empty()){
    auto cur = pq.top();
    pq.pop();
    int v = cur.second;
    if(vis[v])continue;
    vis[v] = true;
    for(auto e : g[v]){
      if(!vis[e.to] && dis[e.to] > dis[v] + e.w){
        dis[e.to] = dis[v] + e.w;
        pq.push(make_pair(dis[e.to] , e.to));
      }
    }
  }
}
```

8.Bellman ford’s algorithm
Shortest path in graph without negative cycle contains at most N – 1nodes.
O(VE) , to check cycle, relax N – 1 times, then relax N-th time, see if any dis[i] Changed, if so, negative cycle exists.

Need to store as a vector of edges for bellman ford.

```
void relax(){
  for(int i = 0; i < edges.size(); ++i){
    int u = edges[i].from;
    int v = edges[i].to;
    int c = edges[i].w;
    if(dis[v] > dis[u] + c){
      dis[v] = dis[u] + c;
    }
  }
}

bool check(){
  for(int i = 1; i <= cnt; ++i){
    dis[i] = inf;
  }
  dis[0] = 0; //assuming starting with node 0
  for(int i = 1; i <= cnt; ++i){
    relax();
  }
  for(int i = 0; i < edges.size(); ++i){
    int u = edges[i].from;
    int v = edges[i].to;
    int c = edges[i].w;
    if(dis[v] > dis[u] + c){
      return false;
    }
  }
  return true;
}
```

9.Floyd-Warshall algorithm
Can compute the shortest distance of all pairs in O(V^3)
(Again, not for negative cycle)
If there is negative cycle, then dis[u][u][n] < 0 and you can't trust the other distances.
dis[i][j][k] = shortest distance from i to j using only the first k nodes.
Then we have dis[i][j][k] = min(dis[i][j][k - 1] , dis[i][k][k - 1] + dis[k][j][k - 1])
Code:

```
#include <iostream>
#include <algorithm>
using namespace std;
const int inf = 1e9;
int N , M;
int dp[500][500];
int main(){
  cin >> N >> M;
  for(int i = 1; i <= N; ++i)
    for(int j = 1; j <= N; ++j)
      dp[i][j] = inf;
  for(int i = 1; i <= M; ++i){
    int x , y , w; cin >> x >> y >> w;
    dp[x][y] = dp[y][x] = w;
  }
  for(int i = 1; i <= N; ++i)dp[i][i] = 0;
```

```
 for(int k = 1; k <= N; ++k)
   for(int i = 1; i <= N; ++i)
     for(int j = 1; j <= N; ++j)
       dp[i][j] = min(dp[i][j] , dp[i][k] + dp[k][j]);
}
```

10. Binary flip
For special graph, such as tree, if want to compute the minimal distance between x and y, how? (non-negative weight on edges)
Observe that, ans(x , y) = dis(x) + dis(y) – 2 * dis(lca(x , y)) where lca(x , y) is the lowest common ancestor of x and y. Where dis(x) is the minimal weighted distance from root to x.
Now the key is to find lca(x , y) fast enough.
Let up[v][i] = the ancestor that is 2^i higher than v
We have up[v][i] = up[up[v][i - 1]][i - 1]
And up[v][0] = v's parent

```
int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}
```

Example problem:
Codeforces-609E-Minimum spanning tree for each edge
Problem statement:
For each edge in an undirected graph, compute the weight of minimum spanning tree that contains this edge.
Solution:
First, build the minimum spanning tree for the graph. Then, each time we trying to include an edge (x , y) , we must introduce a cycle, thus, we need to remove the edge with largest weight on their way to their lca. Time complexity O(ElogE + ElogV)
Code:

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <cmath>
#include <string.h>
#include <cstdio>
using namespace std;
typedef long long ll;
typedef pair<int , int> ii;

```

```cpp
const int maxn = 2e5 + 10;
const ll inf = 1e18;
int N , M;
vector<ii> g[maxn]; //minimum spanning tree
ll ans[maxn];
int a[maxn];
int up[maxn][22];
ll MAX[maxn][22];
ll minval;
int L , timmer , tin[maxn] , tout[maxn];
struct Edge{
  int x , y;
  ll w;
  int id;
  Edge(){}
  Edge(int x_ , int y_ , ll w_ , int id_) : x(x_) , y(y_) , w(w_) , id(id_) {}
  bool operator < (const Edge& other) const {
    return w < other.w;
  }
} edges[maxn];

int find(int x){
  return a[x] == x ? x : a[x] = find(a[x]);
}

void add(int x , int y){
  x = find(x);
  y = find(y);
  if(x != y)
    a[x] = y;
}

void MST(){
  minval = 0;
  vector<int> ids;
  for(int k = 0; k < M; ++k){
    int i = edges[k].x , j = edges[k].y;
    if(find(i) != find(j)){
      add(i , j);
      ids.push_back(edges[k].id);
      minval += edges[k].w;
      g[edges[k].x].emplace_back(edges[k].y , edges[k].w);
      g[edges[k].y].emplace_back(edges[k].x , edges[k].w);
    }
  }
  for(int i : ids)ans[i] = minval;
}

void DFS(int v , int p , ll cost){
  tin[v] = ++timmer;
  up[v][0] = p;
  MAX[v][0] = cost;
  for(int i = 1; i <= L; ++i){
    up[v][i] = up[up[v][i - 1]][i - 1];
    MAX[v][i] = max(MAX[v][i - 1] , MAX[up[v][i - 1]][i - 1]);
  }
  for(auto& e : g[v]){
    if(e.first != p){
      DFS(e.first , v , e.second);
    }
  }
  tout[v] = ++timmer;
}

void preprocess(){
  L = ceil(log2(N));
  for(int i = 1; i <= N; ++i)
    for(int j = 0; j <= L; ++j)
      MAX[i][j] = up[i][j] = 0;
  timmer = 0;
  DFS(1 , 1 , 0);
}

bool isAncestor(int u , int v){ //u is v's ancestor
  return tin[u] <= tin[v] && tout[u] >= tout[v];
}

ll solve(int x , int y){ //return the largest edge on their way to their lca
  ll ret = 0;
  if(isAncestor(x , y)){
    for(int i = L; i >= 0; --i){
      if(!isAncestor(up[y][i] , x)){
        ret = max(ret , MAX[y][i]);
        y = up[y][i];
      }
    }
    ret = max(ret , MAX[y][0]);
  }
  else if(isAncestor(y , x)){
    for(int i = L; i >= 0; --i){
      if(!isAncestor(up[x][i] , y)){
        ret = max(ret , MAX[x][i]);
        x = up[x][i];
      }
    }
    ret = max(ret , MAX[x][0]);
  }
  else{
    int u = x , v = y;
    for(int i = L; i >= 0; --i){
      if(!isAncestor(up[u][i] , y)){
        ret = max(ret , MAX[u][i]);
        u = up[u][i];
      }
    }
    ret = max(ret , MAX[u][0]);
    for(int i = L; i >= 0; --i){
      if(!isAncestor(up[v][i] , x)){
        ret = max(ret , MAX[v][i]);
        v = up[v][i];
      }
    }
    ret = max(ret , MAX[v][0]);
  }
  return ret;
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
```

```
    cin >> N >> M;
    for(int i = 0; i < M; ++i){
      cin >> edges[i].x >> edges[i].y >> edges[i].w;
      edges[i].id = i;
      ans[i] = inf;
    }
    for(int i = 1; i <= N; ++i)a[i] = i;
    sort(edges , edges + M);
    MST();
    preprocess();
    for(int i = 0; i < M; ++i){
      if(ans[edges[i].id] >= inf){
        ans[edges[i].id] = minval - solve(edges[i].x , edges[i].y) +
edges[i].w;
      }
    }
    for(int i = 0; i < M; ++i){
      cout << ans[i] << endl;
      //printf("%lld\n" , ans[i]);
    }
  }
```

Max flow

Dinics:
Time complexity: O(V * V * E)
For unit graph: O(E * sqrt(V))
The key is to construct flow graph!

Template:
Example problem:
Magic Potion

Code:
```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
#include <queue>
#include <string.h>
#include <cmath>
//#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int maxn = 1500;
const ll INF = 1e12;
const double ep = 1e-8;
int N , M , K;
vector<pair<int , int>> E;

struct Edge {
  int from, to;
  ll cap, flow;
  Edge(int u, int v, ll c, ll f) : from(u), to(v), cap(c), flow(f) {}
};

struct Dinic {
  int n, m, s, t;
  vector<Edge> edges;
  vector<int> G[maxn];
  int d[maxn], cur[maxn];
  bool vis[maxn];

  void init(int n) {
    for (int i = 0; i < maxn; i++) G[i].clear();
    edges.clear();
  }

  void add(int from, int to, ll cap) {
    edges.push_back(Edge(from, to, cap, 0));
    edges.push_back(Edge(to, from, 0 , 0));
    m = edges.size();
    G[from].push_back(m - 2);
    G[to].push_back(m - 1);
  }

  bool BFS() {
    memset(vis, 0, sizeof(vis));
    queue<int> Q;
    Q.push(s);
    d[s] = 0;
    vis[s] = 1;
    while (!Q.empty()) {
      int x = Q.front();
      Q.pop();
      for (int i = 0; i < G[x].size(); i++) {
        Edge& e = edges[G[x][i]];
        if (!vis[e.to] && e.cap - e.flow > ep) {
          vis[e.to] = 1;
          d[e.to] = d[x] + 1;
          Q.push(e.to);
        }
      }
    }
    return vis[t];
  }

  ll DFS(int x, ll a) {
    if (x == t || a == 0) return a;
    ll flow = 0, f;
    for (int& i = cur[x]; i < G[x].size(); i++) {
      Edge& e = edges[G[x][i]];
      if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap - e.flow))) > 0) {
        e.flow += f;
        edges[G[x][i] ^ 1].flow -= f;
        flow += f;
        a -= f;
        if (a == 0) break;
      }
    }
    return flow;
  }

  ll maxFlow(int s, int t) {
    this->s = s;
    this->t = t;
    double flow = 0;
    while (BFS()) {
      memset(cur, 0, sizeof(cur));
      flow += DFS(s, INF);
    }
    return flow;
  }

  void DFS2(int v){
    vis[v] = true;
    for(int i = 0; i < G[v].size(); ++i){
      Edge e = edges[G[v][i]];
      if(!vis[e.to] && e.cap - e.flow > ep){
        DFS2(e.to);
      }
    }
  }

};

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> N >> M >> K;
  Dinic g;
  int src = 0 , portion = N + M + 1 , sink = N + M + 2;
  g.init(N + M + 3);
  g.add(src , portion , K);
  for(int i = 1; i <= N; ++i){
    g.add(src , i , 1);
    g.add(portion , i , 1);
    int t; cin >> t;
    for(int j = 1; j <= t; ++j){
      int monster; cin >> monster;
      g.add(i , monster + N , 1);
    }
  }
  for(int i = 1; i <= M; ++i){
    g.add(i + N , sink , 1);
  }
  cout << g.maxFlow(src , sink) << endl;
  return 0;
}
```

Example problem:
Given an undirected graph, each edge with weight, and a sink and a source, determine if the minimal capacity cut of this graph is unique.

Solution:
Run Dinic algorithm first, in the residual network, mark all vertices that are reachable from source and the vertices that can reach sink. If this all vertices are marked then the min cut is unique otherwise not. (Can reach means for some edge, it's flow is less than it's capacity)

Code:

(省去 max flow 的部分，只保留 DFS 部分)
```
  void dfs1(int v){
    vis[v] = true;
    for(int i = 0; i < G[v].size(); ++i){
      Edge& e = edges[G[v][i]];
      if(!vis[e.to] && e.cap > e.flow){
        dfs1(e.to);
      }
    }
  }

  void dfs2(int v){
    vis[v] = true;
    for(int i = 0; i < G[v].size(); ++i){
      Edge& e = edges[G[v][i] ^ 1];
    if(!vis[e.from] && e.cap > e.flow){
      dfs2(e.from);
    }
  }
}

bool uniq_cut(int s , int t){
  memset(vis , 0 , sizeof(vis));
  dfs1(s);
  dfs2(t);
  for(int i = 1; i <= n; ++i){
    if(!vis[i])return false;
  }
  return true;
}
```

DP optimization

1.Convex hull trick
Where to apply?
Eg. dp[i] = min{dp[j] + m[j] * p[i] | j < i}
Where m[j] decreases when j increases.
Let p[x] values be x-axis, DP[x] values be y -axis
If we represent each DP[i] as lines in this plane, then we find that, the optimal DPs
Are the upper convex hull. (so, we should have a data structure that keeps the line’s gradient in decreasing order)
Observe each x coordinate of intersection(line n , line n + 1) keeps moving to the right.
When adding a line, we need to check it’s intersection with the last line to decide if we want to keep the last line or not.
When query x for optimal value, we can binary search to find the line that dominant the range that contains x.

Code with example problem:
Codeforces-319C cutting tree
Problem statement in short:
Height: a[1] < a[2] <…< a[N] , cost: b[1] > b[2] > … > b[N]
a[1] = 1, b[N] = 0, each time cutting one unit need to charge, where charge cost is some b[i] per unit where i is the largest index of the completely cut tree.
Find min cost to cut all N trees completely.

Solution:
2 greedy observation:
1.Each time we cut a tree, we must cut it completely.
2.Each time we cut a tree i, the next tree we cut must have index strictly higher than i.
Then we have dp[i] = "minimal cost to cut the last i trees given that the N – i + 1 th tree is cut"
dp[i] = min{dp[j] + a[N – i + 1] * b[N – j + 1] | j < i}
let gradient be a[N – i + 1] and intersecting point be dp[i]
gradient decreases as i increases and this is a minimization problem, thus convex hull trick is applicable.
Code:
```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
typedef long long ll;

struct line{
  ll m , b;
};

double intersect(line x , line y){
  return (double)(y.b - x.b) / (x.m - y.m);
}

const int maxn = 1e5 + 10;
const ll inf = 1e8;
ll a[maxn] , b[maxn];
ll dp[maxn];
//dp[i] = minimal cost to cut the last i trees given that the last ith tree has
been cut
//dp[i] = min{dp[j] + a[N - j + 1] * b[N - i + 1]}
vector<line> cht;
int N;

void add(line l){
  int n = cht.size();
  while(n >= 2 && intersect(cht[n - 1] , cht[n - 2]) >= intersect(cht[n - 1] , l)){
    cht.pop_back();
    --n;
  }
  cht.push_back(l);
}

ll query(ll x){
  int l = 0 , r = cht.size() - 2;
  int ret = cht.size() - 1;
  while(l <= r){
    int mid = (l + r) >> 1;
    if(intersect(cht[mid] , cht[mid + 1]) >= x)
      ret = mid , r = mid - 1;
    else
```

```
      l = mid + 1;
  }
  return cht[ret].m * x + cht[ret].b;
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> N;
  for(int i = 1; i <= N; ++i)cin >> a[i];
  for(int i = 1; i <= N; ++i)cin >> b[i];
  dp[1] = 0;
  line l1;
  l1.m = a[N];
  l1.b = dp[1];
  add(l1);
  for(int i = 2; i <= N; ++i){
    dp[i] = query(b[N - i + 1]);
    line li;
    li.m = a[N - i + 1];
    li.b = dp[i];
    add(li);
  }
  cout << dp[N] << endl;
}
```

Maximisation:

Example problem:

Codeforces-1083E-The fair Nut and Rectangle.

Problem statement:

You are given n rectangles with vertexes in $(0,0)$, $(x_i,0)$ (x_i,y_i), $(0,y_i)$. For each rectangle, you are also given a number a_i. Choose some of them that the area of union minus sum of a_i of the chosen ones is maximum.

It is guaranteed that there are no nested rectangles.

Solution:

No nested rectangles implies that if we sort rectangle by their top right points, then if we sweep from small x to large x , their y decrease.

Let DP[i] = "maximal value if we choose a set of rectangles up to the i-th one, and i-th rectangle is included in the set".

$$DP[i] = max\{DP[j] + y[j] * (x[i] - x[j]) \mid j < i\} - a[i]$$

Which is $DP[i] = max\{DP[j] - x[j] * y[i]\} + y[i] * x[i] - a[i]$

Since this is a maximisation problem, we want the gradient to be increasing, x[i] increases so It can be gradient, and intersection is DP[i].

Code:

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
typedef long long ll;
typedef pair<int , int> ii;

const int maxn = 1e6 + 10;
int N;
ll dp[maxn];
```

```
struct line{
  ll m , b;
  line(){}
  line(ll m_ , ll b_) : m(m_) , b(b_) {}
};

double intersect(line a , line b){
  return (double)(b.b - a.b) / (a.m - b.m);
}

vector<line> cht;

void add(line l){
  int n = cht.size();
  while(n > 1 && intersect(cht[n - 1] , cht[n - 2]) >= intersect(cht[n - 1] , l)){
    cht.pop_back();
    n = cht.size();
  }
  cht.push_back(l);
}

ll query(ll x){
  int l = 0 , r = cht.size() - 2 , best = cht.size() - 1;
  //find largest best such that x <= intersect(cht[best] , cht[best + 1])
  while(l <= r){
    int mid = (l + r) >> 1;
    if(x <= intersect(cht[mid] , cht[mid + 1])){
      best = mid;
      r = mid - 1;
    }
    else{
      l = mid + 1;
    }
  }
  return cht[best].m * x + cht[best].b;
}

struct rec{
  ll x , y , a;
} R[maxn];

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> N;
  for(int i = 1; i <= N; ++i)
    cin >> R[i].x >> R[i].y >> R[i].a;
  sort(R + 1 , R + 1 + N , [&](rec& a , rec& b){
    return a.x < b.x;
  });
  line l0; l0.m = l0.b = 0; add(l0);
  ll ret = 0;
  for(int i = 1; i <= N; ++i){
    dp[i] = query(-R[i].y) - R[i].a + (ll)R[i].x * R[i].y;
    ret = max(ret , dp[i]);
    line l;
    l.m = R[i].x;
    l.b = dp[i];
    add(l);
  }
  cout << ret << endl;
}
```

Say dp[i] = min/max of dp[j]…

For intersection point, we want it to be a function in terms of j, and gradient in terms of j as well. But usually, gradient is chosen such that it is a product with some function in terms of i.

2.Divide and conquer

$$DP[k][i] = min\{DP[k - 1][j] + Cost(j, i) \mid j < i\}$$

And $Cost(j, i)$ is convex, then we can use this trick. (derivative is increasing)
(More formally, the cost satisfy the quadrangle inequality

$$a < b < c < d \; and \; cost(a, d) - cost(b, d) >= cost(a, c) - cost(b, c)$$

Which can be represented as $(B + c)^2 - B^2 \geq (A + c)^2 - A^2$, it holds when B > A >= 0
Which is true when cost is convex.)

Say the best j for DP[k][i] is opt[k][i], then we know opt[k][i] <= opt[k][i + 1]
So, if we know opt[k][i - 1] and opt[k][i + 1] then to find opt[k][i] we only need to consider range [opt[k][i-1] , opt[k][i+1]].
So, to compute DP[k][1…N], we start finding DP[k][N/2] then for i < N/2, we only consider range [1..opt[k][N/2]]. For i > N/2, we only consider range [opt[k][N/2]…N]. And thus divide and conquer.

Example problem:
IOI-Guardians of the Lunatics
Problem statement:
Minimise some sums.
Solution:
Easy DP equation:
DP[k][i] = min{DP[k - 1][j] + (i – j + 1) * (C[j + 1] + … C[i])} -> O(K*N*N)
Since C[i] >= 0, Cost(i , j) is convex (keep increasing), therefore D & C. -> O(K * N * logN)
Code:

```
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long ll;

const int maxn = 8e3 + 10;
const int maxk = 8e2 + 10;
const ll inf = 1e18;
ll dp[maxk][maxn];
ll sum[maxn] , C[maxn];

int N , K;

//cost of segment (j , i]

ll cost(int i , int j){
  return (ll)(i - j) * (sum[i] - sum[j]);
}
```

```
//query range [l , r] , solve for dp[k][x , y)

void solve(int l , int r , int x , int y , int k){
  if(x >= y)return;
  int best = -1;
  int mid = (x + y) >> 1;
  dp[k][mid] = inf;
  for(int i = l; i <= min(r , mid); ++i){ // be careful here, search range won't go over mid
    ll val = dp[k - 1][i] + cost(mid , i);
    if(val < dp[k][mid])dp[k][mid] = val , best = i;
  }
  solve(l , best , x , mid, k);
  solve(best , r , mid + 1 , y , k);
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> N >> K;
  sum[0] = 0;
  for(int i = 1; i <= N; ++i)cin >> C[i] , sum[i] = sum[i - 1] + C[i];
  for(int i = 1; i <= N; ++i)dp[1][i] = cost(i , 0);
  for(int k = 2; k <= K; ++k){
    solve(0 , N , 1 , N + 1 , k);
  }
  cout << dp[K][N] << endl;
}
```

If we have a maximisation problem, we want the opposite, namely,
cost(a,d)-cost(b,d)<=cost(a,c)-cost(b,c)
which means, it is harder to increase the cost of larger segments. Eg, sqrt() , log()
again, opt[k][i]<=opt[k][i+1]

Maths

1.Fast power

$$x^n = (x^{n/2})^2$$

Code:

```
ll qpow(ll x , ll n){
  if(n <= 0)return 1;
  ll ret = qpow(x , n / 2);
  ret = ret * ret;
  if(n % 2 == 1)ret *= x;
  return ret;
}
```

2.Fermat's little theorm

$$a^{m-1}\ (mod\ m) = 1$$

For prime m.

So we get $a^{m-2} = a^{-1}\ (mod\ m)$

3.Prime finding.

Check if a number is prime $O(sqrt(N))$

Find all primes in a range -> Sieve algorithm

$O\left(\frac{N}{1}+\frac{N}{2}+\cdots+\frac{N}{N}\right)$ = $O(NlogN)$

Code:

Find all primes in [1,N]

```
for(int i = 2; i < N; ++i){
   if(!vis[i]){
     primes.push_back(i);
     for(int j = i; j < N; j += i){
       vis[j] = true;
     }
   }
 }
```

Prime factorisation for each number in [1,N]

```
#include <iostream>
#include <vector>
using namespace std;

const int maxn = 1e5;
bool vis[maxn];
vector<int> fact[maxn];
int N;

int main(){
  cin >> N;
  for(int i = 2; i <= N; ++i){
    if(!vis[i]){
      fact[i].push_back(i);
      for(int j = 2 * i; j <= N; j += i){
        vis[j] = true;
        int tmp = j;
        while(tmp % i == 0){
          fact[j].push_back(i);
          tmp /= i;
        }
      }
    }
```

```
 }
 for(int i = 2; i <= N; ++i){
   cout << i << " fac : ";
   for(int j : fact[i])cout << j << " ";
   cout << endl;
 }
}
```

4.GCD

```
int gcd(int a , int b){
        return b ? gcd(b , a % b) : a;
}
```

5.Extended Euclidean algorithm

Find x and y for ax + by = gcd(a , b)

```
int gcd ( int a, int b, int & x, int & y) {
 if(a == 0){
   x = 0; y = 1;
   return b;
 }
 int x1 , y1;
 int d = gcd(b % a , a , x1 , y1);
 x = y1 - (b / a) * x1;
 y = x1;
 return d;
}
```

6.Matrix

Example problem:

Codeforces-1117D

DP[i] = DP[i - 1] + DP[i - M]

We use matrix to speed it up.

```
#include <iostream>
using namespace std;
typedef long long ll;

const ll mod = 1e9 + 7;
ll N , M;

struct Matrix{
  static const int maxn = 105;
  ll A[maxn][maxn];
  int n;

  Matrix(int n_): n(n_) {
    for(int i = 0; i < n; ++i)
      for(int j = 0; j < n; ++j)
        A[i][j] = 0;
  }

  Matrix operator * (const Matrix& o) const {
    Matrix ret(n);
    for(int i = 0; i < n; ++i)
      for(int j = 0; j < n; ++j)
```

```
        for(int k = 0; k < n; ++k){
          ll c = A[i][k] * o.A[k][j] % mod;
          ret.A[i][j] = (ret.A[i][j] + c) % mod;
        }
    return ret;
  }

  static Matrix identity(int n){
    Matrix ret(n);
    for(int i = 0; i < n; ++i)
      ret.A[i][i] = 1;
    return ret;
  }

  Matrix operator ^ (long long k) const {
    Matrix ret = identity(n);
    Matrix a = *this;
    while(k){
      if(k & 1)ret = ret * a;
      a = a * a;
      k /= 2;
    }
    return ret;
  }

};

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> N >> M;
  if(N < M){
    cout << 1 << endl;
    return 0;
  }
  Matrix a(M);
  a.A[0][0] = a.A[0][M - 1] = 1;
  for(int i = 1; i < M; ++i){
    a.A[i][i - 1] = 1;
  }
  a = a ^ (N - M + 1);
  ll ret = 0;
  for(int i = 0; i < M; ++i){
    ret = (ret + a.A[0][i]) % mod;
  }
  cout << (ret + mod) % mod << endl;
}
```

Things to keep in mind

Given a problem, rather than seating there doing nothing, better try different stuffs.
Start with simple cases, think about under what conditions the solution is easy to get,
See if the solution has monotonicity, if so, maybe we can do binary search over solution space. Does the order matter? If not maybe we can try to solve the problem in some certain ordering. Think of a brute force solution first, then see if we can use data structure to optimize it.
If the problem seems hard even for brute force, try Max flow.

The following is some important things to know:

Shortest distance between a and b in a tree = dis(a) + dis(b) – 2 * dis(lca(a,b))
Given that each edge has positive weight.
Where dis(a) is just the distance from root to a

Good way to code binary search:
(find smallest x such that f(x) is true)

```
Int l = 0 , r = N , best = -1;
While(l <= r){
        Int mid = (l + r) >> 1;
        If(check(mid)){
                best = mid;
                r = mid – 1;
        }
        Else{
                l = mid + 1;
        }
}
```

Read from files / write to files:

```
  freopen("xxx.in" , "r" , stdin);
  freopen("xxx.out" , "w" , stdout);
…..use scanf / printf
  fclose(stdin);
  fclose(stdout);
```