Codebook

March 20, 2023

Contents

1 Setup

1.1 Template

```cpp
#include <bits/stdc++.h>
#include <bits/extc++.h>
#define F first
#define S second
#define pb push_back
#define pob pop_back
#define pf push_front
#define pof pop_front
#define mp make_pair
#define mt make_tuple
#define all(x) (x).begin(),(x).end()
using namespace std;
//using namespace __gnu_pbds;
using pii = pair<long long,long long>;
using ld = long double;
using ll = long long;
const int mod = 1000000007;
const int mod2 = 998244353;
const ld PI = acos(-1);
#define Bint __int128
#define int long long
```

1.2 vimrc

```vim
syntax on
set mouse=a
set nu
set ts=4
set sw=4
set smartindent
set cursorline
set hlsearch
set incsearch
set t_Co=256
nnoremap y ggyG
colorscheme afterglow
au BufNewFile *.cpp 0r ~/default_code/default.cpp |
↪ let IndentStyle = "cpp"
```

2 Data-structure

2.1 PBDS

```cpp
gp_hash_table<T, T> h;
tree<T, null_type, less<T>, rb_tree_tag,
↪ tree_order_statistics_node_update> tr;
tr.order_of_key(x); // find x's ranking
tr.find_by_order(k); // find k-th minimum, return
↪ iterator
```

2.2 LazyTagSegtree

```cpp
struct segment_tree{
  int seg[N << 2];
  int tag1[N << 2], tag2[N << 2];
  void down(int l, int r, int idx, int pidx){
```

```
    int v = tag1[pidx], vv = tag2[pidx];
    if(v)
      tag1[idx] = v, seg[idx] = v * (r - l + 1), tag2[idx] = 0;
    if(vv)
      tag2[idx] += vv, seg[idx] += vv * (r - l + 1);
  }
  void Set(int l, int r, int ql, int qr, int v, int idx = 1){
    if(ql == l && qr == r){
      tag1[idx] = v;
      tag2[idx] = 0;
      seg[idx] = v * (r - l + 1);
      return;
    }
    int mid = (l + r) >> 1;
    down(l, mid, idx << 1, idx);
    down(mid + 1, r, idx << 1 | 1, idx);
    tag1[idx] = tag2[idx] = 0;
    if(qr <= mid)
      Set(l, mid, ql, qr, v, idx << 1);
    else if(ql > mid)
      Set(mid + 1, r, ql, qr, v, idx << 1 | 1);
    else{
      Set(l, mid, ql, mid, v, idx << 1);
      Set(mid + 1, r, mid + 1, qr, v, idx << 1 | 1);
    }
    seg[idx] = seg[idx << 1] + seg[idx << 1 | 1];
  }
  void Increase(int l, int r, int ql, int qr, int v, int idx = 1){
    if(ql ==l && qr == r){
      tag2[idx] += v;
      seg[idx] +=  v * (r - l + 1);
      return;
    }
    int mid = (l + r) >> 1;
    down(l, mid, idx << 1, idx);
    down(mid + 1, r, idx << 1 | 1, idx);
    tag1[idx] = tag2[idx] = 0;
    if(qr <= mid)
      Increase(l, mid, ql, qr, v, idx << 1);
    else if(ql > mid)
      Increase(mid + 1, r, ql, qr, v, idx << 1 | 1);
    else{
      Increase(l, mid, ql, mid, v, idx << 1);
      Increase(mid + 1, r, mid + 1, qr, v, idx << 1 | 1);
    }
    seg[idx] = seg[idx << 1] + seg[idx << 1 | 1];
  }
  int query(int l, int r, int ql, int qr, int idx = 1){
    if(ql ==l && qr == r)
      return seg[idx];
    int mid = (l + r) >> 1;
    down(l, mid, idx << 1, idx);
    down(mid + 1, r, idx << 1 | 1, idx);
    tag1[idx] = tag2[idx] = 0;
    if(qr <= mid)
      return query(l, mid, ql, qr, idx << 1);
    else if(ql > mid)
      return query(mid + 1, r, ql, qr, idx << 1 | 1);
    return query(l, mid, ql, mid, idx << 1) + query(mid + 1, r, mid + 1, qr, idx << 1 | 1);
  }
  void modify(int l, int r, int ql, int qr, int v, int type){
    // type 1: increasement, type 2: set
    if(type == 2)
      Set(l, r, ql, qr, v);
    else
      Increase(l, r, ql, qr, v);
  }
```

2.3 LiChaoTree

```
struct line{
  int m, c;
  int val(int x){
    return m * x + c;
  }
  line(){}
  line(int _m, int _c){
    m = _m, c = _c;
  }
};
struct Li_Chao_Tree{
  line seg[N << 2];
  void ins(int l, int r, int idx, line x){
    if(l == r){
      if(x.val(l) > seg[idx].val(l))
        seg[idx] = x;
      return;
    }
    int mid = (l + r) >> 1;
    if(x.m < seg[idx].m)
      swap(x, seg[idx]);
    // ensure x.m > seg[idx].m
    if(seg[idx].val(mid) <= x.val(mid)){
      swap(x, seg[idx]);
      ins(l, mid, idx << 1, x);
    }
    else
      ins(mid + 1, r, idx << 1 | 1, x);
  }
  int query(int l, int r, int p, int idx){
    if(l == r)
      return seg[idx].val(l);
    int mid = (l + r) >> 1;
    if(p <= mid)
      return max(seg[idx].val(p), query(l, mid, p, idx << 1));
    else
      return max(seg[idx].val(p), query(mid + 1, r, p, idx << 1 | 1));
  }
```

2.4 Treap

```
mt19937 mtrd(chrono::steady_clock::now().time_since_epoch().count());
struct Treap{
  Treap *l, *r;
  int pri, key, sz;
  Treap(){}
  Treap(int _v){
    l = r = NULL;
    pri = mtrd();
    key = _v;
    sz = 1;
  }
  ~Treap(){
        if ( l )
            delete l;
        if ( r )
            delete r;
    }
  void push(){
    for(auto ch : {l, r}){
      if(ch){
        // do something
      }
    }
  }
};
int getSize(Treap *t){
  return t ? t->sz : 0;
}
void pull(Treap *t){
  t->sz = getSize(t->l) + getSize(t->r) + 1;
}
Treap* merge(Treap* a, Treap* b){
  if(!a || !b)
    return a ? a : b;
  if(a->pri > b->pri){
    a->push();
    a->r = merge(a->r, b);
    pull(a);
    return a;
  }
  else{
    b->push();
    b->l = merge(a, b->l);
    pull(b);
    return b;
  }
}
void splitBySize(Treap *t, Treap *&a, Treap *&b, int k){
  if(!t)
    a = b = NULL;
  else if(getSize(t->l) + 1 <= k){
    a = t;
    a->push();
    splitBySize(t->r, a->r, b, k - getSize(t->l) - 1);
    pull(a);
  }
  else{
    b = t;
    b->push();
    splitBySize(t->l, a, b->l, k);
    pull(b);
  }
}
void splitByKey(Treap *t, Treap *&a, Treap *&b, int k){
    if(!t)
        a = b = NULL;
    else if(t->key <= k){
        a = t;
        a->push();
        splitByKey(t->r, a->r, b, k);
        pull(a);
    }
    else{
        b = t;
        b->push();
        splitByKey(t->l, a, b->l, k);
        pull(b);
    }
}
// O(n) build treap with sorted key nodes
void traverse(Treap *t){
  if(t->l)
    traverse(t->l);
  if(t->r)
    traverse(t->r);
  pull(t);
}
Treap *build(int n){
  vector<Treap*>st(n);
  int tp = 0;
  for(int i = 0, x; i < n; i++){
    cin >> x;
    Treap *nd = new Treap(x);
    while(tp && st[tp - 1]->pri < nd->pri)
      nd->l = st[tp - 1], tp--;
    if(tp)
      st[tp - 1]->r = nd;
    st[tp++] = nd;
  }
  if(!tp){
    st[0] = NULL;
    return st[0];
  }
  traverse(st[0]);
  return st[0];
}
```

3 Graph

3.1 RoundSquareTree

```
int cnt;
int dep[N], low[N]; // dep == -1 -> unvisited
vector<int>G[N], rstree[2 * N]; // 1 ~ n: round, n + 1 ~ 2n: square
vector<int>stk;
void init(){
    cnt = n;
    for(int i = 1; i <= n; i++){
        G[i].clear();
```

```
        rstree[i].clear();
        rstree[i + n].clear();
        dep[i] = low[i] = -1;
    }
    dep[1] = low[1] = 0;
}
void tarjan(int x, int px){
    stk.push_back(x);
    for(auto i : G[x]){
        if(dep[i] == -1){
            dep[i] = low[i] = dep[x] + 1;
            tarjan(i, x);
            low[x] = min(low[x], low[i]);
            if(dep[x] <= low[i]){
                int z;
        cnt++;
                do{
                    z = stk.back();
                    rstree[cnt].push_back(z);
                    rstree[z].push_back(cnt);
                    stk.pop_back();
                }while(z != i);
                rstree[cnt].push_back(x);
                rstree[x].push_back(cnt);
            }
        }
        else if(i != px)
            low[x] = min(low[x], dep[i]);
    }
}
```

3.2 SCC

```
struct SCC{
  int n;
  int cnt;
  vector<vector<int>>G, revG;
  vector<int>stk, sccid;
  vector<bool>vis;
  SCC(): SCC(0) {}
  SCC(int _n): n(_n), G(_n + 1), revG(_n + 1), sccid(_n + 1), vis(_n + 1), cnt(0) {}
  void addEdge(int u, int v){
    // u -> v
    assert(u > 0 && u <= n);
    assert(v > 0 && v <= n);
    G[u].push_back(v);
    revG[v].push_back(u);
  }
  void dfs1(int u){
    vis[u] = 1;
    for(int v : G[u]){
      if(!vis[v])
        dfs1(v);
    }
    stk.push_back(u);
  }
  void dfs2(int u, int k){
    vis[u] = 1;
    sccid[u] = k;
    for(int v : revG[u]){
      if(!vis[v])
        dfs2(v, k);
    }
  }
  void Kosaraju(){
    for(int i = 1; i <= n; i++)
      if(!vis[i])
        dfs1(i);
    fill(vis.begin(), vis.end(), 0);
    while(!stk.empty()){
      if(!vis[stk.back()])
        dfs2(stk.back(), ++cnt);
      stk.pop_back();
    }
  }
};
```

3.3 2SAT

```
struct two_sat{
  int n;
  SCC G; // u: u, u + n: ~u
  vector<int>ans;
  two_sat(): two_sat(0) {}
  two_sat(int _n): n(_n), G(2 * _n), ans(_n + 1) {}
  void disjunction(int a, int b){
    G.addEdge((a > n ? a - n : a + n), b);
    G.addEdge((b > n ? b - n : b + n), a);
  }
  bool solve(){
    G.Kosaraju();
    for(int i = 1; i <= n; i++){
      if(G.sccid[i] == G.sccid[i + n])
        return false;
      ans[i] = (G.sccid[i] > G.sccid[i + n]);
    }
    return true;
  }
};
```

3.4 bridge

```
int dep[N], low[N];
vector<int>G[N];
vector<pair<int, int>>bridge;
void init(){
  for(int i = 1; i <= n; i++){
    G[i].clear();
    dep[i] = low[i] = -1;
  }
  dep[1] = low[1] = 0;
}
void tarjan(int x, int px){
  for(auto i : G[x]){
    if(dep[i] == -1){
      dep[i] = low[i] = dep[x] + 1;
      tarjan(i, x);
      low[x] = min(low[x], low[i]);
      if(low[i] > dep[x])
        bridge.push_back(make_pair(i, x));
    }
    else if(i != px)
      low[x] = min(low[x], dep[i]);
```

```
  }
}
```

3.5 BronKerbosch$_algorithm$

```
vector<vector<int>>maximal_clique;
int cnt, G[N][N], all[N][N], some[N][N], none[N][N];
void dfs(int d, int an, int sn, int nn)
{
    if(sn == 0 && nn == 0){
    vector<int>v;
    for(int i = 0; i < an; i++)
      v.push_back(all[d][i]);
    maximal_clique.push_back(v);
    cnt++;
    }
  int u = sn > 0 ? some[d][0] : none[d][0];
    for(int i = 0; i < sn; i ++)
    {
        int v = some[d][i];
        if(G[u][v])
      continue;
        int tsn = 0, tnn = 0;
        for(int j = 0; j < an; j ++)
      all[d + 1][j] = all[d][j];
        all[d + 1][an] = v;
        for(int j = 0; j < sn; j ++)
            if(g[v][some[d][j]])
        some[d + 1][tsn ++] = some[d][j];
        for(int j = 0; j < nn; j ++)
            if(g[v][none[d][j]])
        none[d + 1][tnn ++] = none[d][j];
        dfs(d + 1, an + 1, tsn, tnn);
        some[d][i] = 0, none[d][nn ++] = v;
    }
}
void process(){
    cnt = 0;
    for(int i = 0; i < n; i ++)
    some[0][i] = i + 1;
    dfs(0, 0, n, 0);
}
```

3.6 Theorem

- Kosaraju's algorithm visit the strong connected components in topolocical order at second dfs.
- Euler's formula on planar graph: $V - E + F = C + 1$
- Kuratowski's theorem: A simple graph G is a planar graph iff G doesn't has a subgraph H such that H is homeomorphic to K_5 or $K_{3,3}$
- A complement set of every vertex cover correspond to a independent set. $\Rightarrow$ Number of vertex of maximum independent set + Number of vertex of minimum vertex cover $= V$
- Maximum independent set of $G =$ Maximum clique of the complement graph of G .
- A planar graph G colored with three colors iff there exist a maximal clique I such that $G - I$ is a bipartite.

4 String

4.1 RollingHash

```
struct Rolling_Hash{
  int n;
  const int P[5] = {146672737, 204924373, 585761567, 484547929, 116508269};
  const int M[5] = {922722049, 952311013, 955873937, 901981687, 993179543};
  vector<int>PW[5], pre[5], suf[5];
  Rolling_Hash(): Rolling_Hash("") {}
  Rolling_Hash(string s): n(s.size()){
    for(int i = 0; i < 5; i++){
      PW[i].resize(n), pre[i].resize(n), suf[i].resize(n);
      PW[i][0] = 1, pre[i][0] = s[0] - 'a';
      suf[i][n - 1] = s[n - 1] - 'a';
    }
    for(int i = 1; i < n; i++){
      for(int j = 0; j < 5; j++){
        PW[j][i] = PW[j][i - 1] * P[j] % M[j];
        pre[j][i] = (pre[j][i - 1] * P[j] + s[i] - 'a') % M[j];
      }
    }
    for(int i = n - 2; i >= 0; i--){
      for(int j = 0; j < 5; j++)
        suf[j][i] = (suf[j][i + 1] * P[j] + s[i] - 'a') % M[j];
    }
  }
  int _substr(int k, int l, int r) {
    int res = pre[k][r];
    if(l > 0)
      res -= 1LL * pre[k][l - 1] * PW[k][r - l + 1] % M[k];
    if(res < 0)
      res += M[k];
    return res;
  }
  vector<int>substr(int l, int r){
    vector<int>res(5);
    for(int i = 0; i < 5; ++i)
      res[i] = _substr(i, l, r);
    return res;
  }
};
```

4.2 SuffixArray

```
struct Suffix_Array{
  int n, m; // m is the range of s
  string s;
  vector<int>sa, rk, lcp;
  Suffix_Array(): Suffix_Array(0, 0, "") {};
  Suffix_Array(int _n, int _m, string _s): n(_n), m(_m), sa(_n), rk(_n), lcp(_n), s(_s) {}
```

```cpp
  void Sort(int k, vector<int>&bucket, vector<int>&idx, vector<int>&lst){
    for(int i = 0; i < m; i++)
      bucket[i] = 0;
    for(int i = 0; i < n; i++)
      bucket[lst[i]]++;
    for(int i = 1; i < m; i++)
      bucket[i] += bucket[i-1];
    int p = 0;
    // update index
    for(int i = n - k; i < n; i++)
      idx[p++] = i;
    for(int i = 0; i < n; i++)
      if(sa[i] >= k)
        idx[p++] = sa[i] - k;
    for(int i = n - 1; i >= 0; i--)
      sa[--bucket[lst[idx[i]]]] = idx[i];
  }
  void build(){
    vector<int>idx(n), lst(n), bucket(max(n, m));
    for(int i = 0; i < n; i++)
      bucket[lst[i] = (s[i] - 'a')]++;
    for(int i = 1; i < m; i++)
      bucket[i] += bucket[i - 1];
    for(int i = n - 1; i >= 0; i--)
      sa[--bucket[lst[i]]] = i;
    for(int k = 1; k < n; k <<= 1){
      Sort(k, bucket, idx, lst);
      // update rank
      int p = 0;
      idx[sa[0]] = 0;
      for(int i = 1; i < n; i++){
        int a = sa[i], b = sa[i - 1];
        if(lst[a] == lst[b] && a + k < n && b + k < n && lst[a + k] == lst[b + k]);
        else
          p++;
        idx[sa[i]] = p;
      }
      if(p == n - 1)
        break;
      for(int i = 0; i < n; i++)
        lst[i] = idx[i];
      m = p + 1;
    }
    for(int i = 0; i < n; i++)
      rk[sa[i]] = i;
    buildLCP();
  }
  void buildLCP(){
    // lcp[rk[i]] >= lcp[rk[i - 1]] - 1
    int v = 0;
    for(int i = 0; i < n; i++){
      if(!rk[i])
        lcp[rk[i]] = 0;
      else{
        if(v)
          v--;
        int p = sa[rk[i] - 1];
        while(i + v < n && p + v < n && s[i + v] == s[p + v])
          v++;
        lcp[rk[i]] = v;
      }
    }
  }
};
```

5 Flow

5.1 Dinic

```cpp
struct Max_Flow{
  struct Edge{
    int cap, to, rev;
    Edge(){}
    Edge(int _to, int _cap, int _rev){
      to = _to, cap = _cap, rev = _rev;
    }
  };
  const int inf = 1e18+10;
  int s, t; // start node and end node
  vector<vector<Edge>>G;
  vector<int>dep;
  vector<int>iter;
  void addE(int u, int v, int cap){
    G[u].pb(Edge(v, cap, G[v].size()));
    // direct graph
    G[v].pb(Edge(u, 0, G[u].size() - 1));
    // undirect graph
    // G[v].pb(Edge(u, cap, G[u].size() - 1));
  }
  void bfs(){
    queue<int>q;
    q.push(s);
    dep[s] = 0;
    while(!q.empty()){
      int cur = q.front();
      q.pop();
      for(auto i : G[cur]){
        if(i.cap > 0 && dep[i.to] == -1){
          dep[i.to] = dep[cur] + 1;
          q.push(i.to);
        }
      }
    }
  }
  int dfs(int x, int fl){
    if(x == t)
      return fl;
    for(int _ = iter[x] ; _ < G[x].size() ; _++){
      auto &i = G[x][_];
      if(i.cap > 0 && dep[i.to] == dep[x] + 1){
        int res = dfs(i.to, min(fl, i.cap));
        if(res <= 0)
          continue;
        i.cap -= res;
        G[i.to][i.rev].cap += res;
        return res;
      }
      iter[x]++;
    }
    return 0;
  }
  int Dinic(){
    int res = 0;
    while(true){
```

```
      fill(all(dep), -1);
      fill(all(iter), 0);
      bfs();
      if(dep[t] == -1)
        break;
      int cur;
      while((cur = dfs(s, INF)) > 0)
        res += cur;
    }
    return res;
  }
  void init(int _n, int _s, int _t){
    s = _s, t = _t;
    G.resize(_n + 5);
    dep.resize(_n + 5);
    iter.resize(_n + 5);
  }
};
```

6 Math

6.1 FastPow

```
long long qpow(long long x, long long powcnt, long long tomod){
  long long res = 1;
  for(; powcnt ; powcnt >>= 1 , x = (x * x) % tomod)
    if(1 & powcnt)
      res = (res * x) % tomod;
  return (res % tomod);
```

6.2 EXGCD

```
// ax + by = c
// return (gcd(a, b), x, y)
tuple<long long, long long, long long>exgcd(long long a, long long b){
  if(b == 0)
    return make_tuple(a, 1, 0);
  auto[g, x, y] = exgcd(b, a % b);
  return make_tuple(g, y, x - (a / b) * y);
```

6.3 EXCRT

```
long long inv(long long x){ return qpow(x, mod - 2, mod); }
long long mul(long long x, long long y, long long m){
  x = ((x % m) + m) % m, y = ((y % m) + m) % m;
  long long ans = 0;
  while(y){
    if(y & 1)
      ans = (ans + x) % m;
    x = x * 2 % m;
    y >>= 1;
  }
  return ans;
}
pii ExCRT(long long r1, long long m1, long long r2, long long m2){
  long long g, x, y;
  tie(g, x, y) = exgcd(m1, m2);
  if((r1 - r2) % g)
    return {-1, -1};
  long long lcm = (m1 / g) * m2;
  long long res = (mul(mul(m1, x, lcm), ((r2 - r1) / g), lcm) + r1) % lcm;
  res = (res + lcm) % lcm;
  return {res, lcm};
}
void solve(){
  long long n, r, m;
  cin >> n;
  cin >> m >> r; // x == r (mod m)
  for(long long i = 1 ; i < n ; i++){
    long long r1, m1;
    cin >> m1 >> r1;
    if(r != -1 && m != -1)
      tie(r, m) = ExCRT(r m, r1, m1);
  }
  if(r == -1 && m == -1)
    cout << "no solution\n";
  else
    cout << r << '\n';
}
```

6.4 FFT

```
struct Polynomial{
  int deg;
  vector<int>x;
  void FFT(vector<complex<double>>&a, bool invert){
    int a_sz = a.size();
    for(int len = 1; len < a_sz; len <<= 1){
      for(int st = 0; st < a_sz; st += 2 * len){
        double angle = PI / len * (invert ? -1 : 1);
        complex<double>wnow(1), w(cos(angle), sin(angle));
        for(int i = 0; i < len; i++){
          auto a0 = a[st + i], a1 = a[st + len + i];
          a[st + i] = a0 + wnow * a1;
          a[st + i + len] = a0 - wnow * a1;
          wnow *= w;
        }
      }
    }
    if(invert)
      for(auto &i : a)
        i /= a_sz;
  }
  void change(vector<complex<double>>&a){
    int a_sz = a.size();
    vector<int>rev(a_sz);
    for(int i = 1; i < a_sz; i++){
      rev[i] = rev[i / 2] / 2;
      if(i & 1)
        rev[i] += a_sz / 2;
    }
```

```
    for(int i = 0; i < a_sz; i++)
      if(i < rev[i])
        swap(a[i], a[rev[i]]);
  }
  Polynomial multiply(Polynomial const&b){
    vector<complex<double>>A(x.begin(), x.end()), B(b.x.begin(), b.x.end());
    int mx_sz = 1;
    while(mx_sz < A.size() + B.size())
      mx_sz <<= 1;
    A.resize(mx_sz);
    B.resize(mx_sz);
    change(A);
    change(B);
    FFT(A, 0);
    FFT(B, 0);
    for(int i = 0; i < mx_sz; i++)
      A[i] *= B[i];
    change(A);
    FFT(A, 1);
    Polynomial res(mx_sz);
    for(int i = 0; i < mx_sz; i++)
      res.x[i] = round(A[i].real());
    while(!res.x.empty() && res.x.back() == 0)
      res.x.pop_back();
    res.deg = res.x.size();
    return res;
  }
  Polynomial(): Polynomial(0) {}
  Polynomial(int Size): x(Size), deg(Size) {}
};
```

6.5 GeneratingFunctions

- Ordinary Generating Function $A(x) = \sum_{i\ge 0} a_i x^i$
  - $A(rx) \Rightarrow r^n a_n$
  - $A(x) + B(x) \Rightarrow a_n + b_n$
  - $A(x)B(x) \Rightarrow \sum_{i=0}^{n} a_i b_{n-i}$
  - $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} a_{i_1} a_{i_2} \ldots a_{i_k}$
  - $xA(x)' \Rightarrow n a_n$
  - $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^{n} a_i$
- Exponential Generating Function $A(x) = \sum_{i\ge 0} \frac{a_i}{i!} x_i$
  - $A(x) + B(x) \Rightarrow a_n + b_n$
  - $A^{(k)}(x) \Rightarrow a_{n+k}$
  - $A(x)B(x) \Rightarrow \sum_{i=0}^{n} n i a_i b_{n-i}$
  - $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} n i_1, i_2, \ldots, i_k a_{i_1} a_{i_2} \ldots a_{i_k}$
  - $xA(x) \Rightarrow n a_n$
- Special Generating Function
  - $(1+x)^n = \sum_{i\ge 0} n i x^i$
  - $\frac{1}{(1-x)^n} = \sum_{i\ge 0} i n - 1 x^i$

6.6 Numbers

- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups. $S(n,k) = S(n-1,k-1) + kS(n-1,k), S(n,1) = S(n,n) = 1$ $S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i} i^n$ $x^n = \sum_{i=0}^{n} S(n,i)(x)_i$
- Catalan numbers $C_n = \frac{1}{n+1} 2nn = 2nn - 2nn+1$, $\forall n \ge 0$ $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} = \frac{2(2n+1)}{n+2} C_n,\ C_0 = 1$

6.7 Theorem

- Cayley's Formula
  - Given a degree sequence $d_1, d_2, \ldots, d_n$ for each *labeled* vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ spanning trees.
  - Let $T_{n,k}$ be the number of *labeled* forests on n vertices with k components, such that vertex $1, 2, \ldots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.
- Erdős–Gallai theorem A sequence of nonnegative integers $d_1 \ge \cdots \ge d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1+\cdots+d_n$ is even and $\displaystyle\sum_{i-1}^{k} d_i \le k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$ holds for every $1 \le k \le n$.
- Gale–Ryser theorem A pair of sequences of nonnegative integers $a_1 \ge \cdots \ge a_n$ and $b_1, \ldots, b_n$ is bigraphic if and only if $\displaystyle\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\displaystyle\sum_{i=1}^{k} a_i \le \sum_{i=1}^{n} \min(b_i, k)$ holds for every $1 \le k \le n$.
- Flooring and Ceiling function identity
  - $\lfloor \frac{\lfloor \frac{a}{b} \rfloor}{c} \rfloor = \lfloor \frac{a}{bc} \rfloor$
  - $\lceil \frac{\lceil \frac{a}{b} \rceil}{c} \rceil = \lceil \frac{a}{bc} \rceil$
  - $\lceil \frac{a}{b} \rceil \le \frac{a+b-1}{b}$
  - $\lfloor \frac{a}{b} \rfloor \le \frac{a-b+1}{b}$
- Möbius inversion formula
  - $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$
  - $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$
  - $\sum_{d|n}^{n=1} \mu(d) = 1$
  - $\sum_{d|n}^{n\neq 1} \mu(d) = 0$
- Spherical cap
  - A portion of a sphere cut off by a plane.
  - r: sphere radius, a: radius of the base of the cap, h: height of the cap, θ: $\arcsin(a/r)$.
  - Volume $= \pi h^2(3r-h)/3 = \pi h(3a^2+h^2)/6 = \pi r^3(2+\cos\theta)(1-\cos\theta)^2/3$.
  - Area $= 2\pi rh = \pi(a^2+h^2) = 2\pi r^2(1-\cos\theta)$.