Team notebook

Enrique Ortiz

August 7, 2018

Contents

1 BFS

```
struct Node {
    vector<int> adj;
    int identifier;
    bool isVisited = false;
};

void BFS(Node& Start, vector<Node>& Nodes) {

    Start.isVisited = true;
    queue<Node> Q;
    Q.push(Start);
    while(!Q.empty()) {

        Node V = Q.front();
        Q.pop();

        for(int& k : V.adj) {
       // cerr << V.identifier << ": checking node " << k << endl;
            if (Nodes[k].isVisited == false) {
                Nodes[k].isVisited = true;
                Q.push(Nodes[k]);
                cerr << "Pushed node " << k << endl;
            }
        }
    }
}
```

2 BigInteger

```
typedef long long Long;
struct BigInteger
{
        static const int FACTOR = 1000000000;
        static const int DSZ = 9;
        vector<Long> V;
        BigInteger(Long n = 0){
                do
                {
                        V.push_back(n % FACTOR);
                        n /= FACTOR;
                }while(n > 0);
```

```
}
BigInteger operator<<(const int n)const{
	BigInteger ret;
	int m = n % DSZ , d = n/ DSZ;
	vector<Long> newV( V.size() + d + 2 , 0 );
	for(int i = 0; i < V.size(); ++i){
		newV[i+d] = V[i];
		for(int j = 0; j < m; ++j)
			newV[i] *= 10;
	}
	ret.V = newV;
	ret.adjust();
	return ret;
}
BigInteger sMult(const BigInteger &B)const{
	BigInteger ret;
	ret.V = vector<Long>(V.size() + B.V.size() , 0);
	for(int i = 0; i < V.size(); ++i){
		for(int j = 0; j < B.V.size(); ++j){
			ret.V[i+j] += B.V[j] * V[i];
		}
		for(int j = i; j < ret.V.size(); ++j)
		{
			Long d = ret.V[j] / FACTOR;
			if(j+1 < ret.V.size()) ret.V[j+1] += d;
			else if(d)
			{
				ret.V.push_back(d);
			}
			ret.V[j] %= FACTOR;
		}
	}
	while(ret.V.back() == 0)
		ret.V.pop_back();
	return ret;
}
BigInteger operator*(const BigInteger &B)const{
	if(B.V.size() <= 200 || V.size() <= 200)
		return (*this).sMult(B);
	int SZ = max(V.size()/2 , B.V.size()/2);
	SZ = min(SZ , (int)B.V.size() - 1);
	SZ = min(SZ , (int)V.size() - 1);
	vector<Long> vA1(V.begin(),V.begin()+SZ);
	vector<Long> vA2(V.begin()+SZ,V.end());
	vector<Long> vB1(B.V.begin(),B.V.begin()+SZ);
	vector<Long> vB2(B.V.begin()+SZ,B.V.end());
	BigInteger A1,A2,B1,B2;
	A1.V = vA1;
	A2.V = vA2;
	B1.V = vB1;
	B2.V = vB2;
	BigInteger Z2 = (A2*B2);
	BigInteger Z0 = (A1*B1);
	BigInteger Z1 = ((A1+A2) * (B1+B2)) - Z2 - Z0;
	return (((Z2<<(SZ*2*DSZ)) + (Z1<<(SZ*DSZ))) + Z0);
}
BigInteger operator+(const BigInteger &B)const{
	BigInteger ret;
	ret.V = vector<Long>(max(V.size() , B.V.size()) + 1,0);
	for(int i = 0; i < max(V.size(),B.V.size()); ++i){
		if(i < V.size())ret.V[i] += V[i];
		if(i<B.V.size())ret.V[i] += B.V[i];
	}
	ret.adjust();
	return ret;
}
BigInteger operator-(const BigInteger &B)const{
	BigInteger ret;
	ret.V = vector<Long>(max(V.size() , B.V.size()) + 1,0);
	int c = 0;
	for(int i = 0; i < max(V.size(),B.V.size()); ++i){
		Long d = V[i] - c;
		if(i < B.V.size())d -= B.V[i];
		if(d < 0)d += FACTOR , c = 1;
		else c = 0;
		ret.V[i] = d;
	}
	ret.adjust();
	return ret;
}
Long rawDivide(const vector<Long> &A,const vector<Long> &B)const{
	Long _A = 0 , _B = 0;
	int eq = 0;
	if(A.size() == B.size())
		_A = A.back() , _B = B.back() , eq = 1;
	else if(A.size() > B.size())
		_A = A.back() * FACTOR + A[A.size() - 2] , _B =
		    B.back();
	else
		return 0;
```

```
        if(B.size() > 1){
                char AA[12],BB[12];
                sprintf(AA,"%0*lld",DSZ,A[A.size()-3+eq]);
                sprintf(BB,"%0*lld",DSZ,B[B.size()-2]);
                int idx = 0;
                while(_B < FACTOR / 10){
                        _A = 10 * _A + (AA[idx]-'0');
                        _B = 10 * _B + (BB[idx]-'0');
                        idx++;
                }
        }
        return max( 0LL , _A / _B - 1 );
}
pair<BigInteger,BigInteger> divide(const BigInteger &B)const{
        BigInteger Q;
        BigInteger R;
        for(int i = V.size()-1; i>=0 ; --i)
        {
                R = R << DSZ;
                Q = Q << DSZ;
                R = R + V[i];
                Long rawFactor = rawDivide(R.V,B.V);
                Q.V[0] += rawFactor;
                R = R - (B * rawFactor);
                int cnt = 0;
                while( !(R < B) ){
                        R = R - B;
                        cnt++;
                        Q.V[0] += 1;
                }
        }
        return pair<BigInteger,BigInteger>(Q,R);
}
BigInteger operator/(const BigInteger &B)const{
        return divide(B).first;
}
BigInteger operator%(const BigInteger &B)const{
        return divide(B).second;
}
int operator<(const BigInteger &B)const{
        return (V.size() != B.V.size() ? V.size() < B.V.size() :
            V.back() < B.V.back());
}
int operator>(const BigInteger &B)const{
        return B < (*this);
}
        int digits()const{
                if(V.size() == 0 && V.back() == 0)return 1;
                return (V.size()-1) * DSZ + log10(V.back()) + 1;
        }
        void adjust(){
                for(int i = 0; i < V.size(); ++i)
                {
                        Long d = V[i] / FACTOR;
                        if(i+1 < V.size()) V[i+1] += d;
                        else if(d) V.push_back(d);
                        V[i] %= FACTOR;
                }
                while(V.size() > 1 && V.back() == 0)V.pop_back();
        }
        string str()const{
                string ret = "";
                for(int i = (int)V.size()-1 ; i >= 0 ; --i)
                {
                        char num[12];
                        sprintf(num,"%0*lld",(i+1==V.size()?0:DSZ),V[i]);
                        int n = strlen(num);
                        ret += num;
                }
                return ret;
        }

};
ostream& operator<<(ostream &o,const BigInteger &B){
        return o << B.str();
}
```

3 DisjointSet

```
struct DisjointSet {
        vector<int> P; // if < 0 then negative size, else parentId
        DisjointSet(int N) : P(N, -1) {}
        int find(int x) {
                return P[x] < 0 ? x : (P[x] = find(P[x]));
        }
        bool join(int x,int y) {
```

```
            if((x = find(x)) == (y = find(y))) return false;
            if(P[y] < P[x]) swap(x,y);
            P[x] += P[y];
            P[y] = x;
            return true;
        }
};
```

4 EventProcessing

```
struct Event {
    int x;
    char type;
};

int N;
int L[100004], R[100004];

void solve() {
    for(int i = 0; i < N; i++)
        cin >> L[i] >> R[i];

    vector<Event> events;

    for(int i = 0; i < N; i++) {
        events.push_back({L[i], 'E'});
        events.push_back({R[i], 'X'});
    }

    sort(events.begin(), events.end(),
         [&] (Event a, Event b) -> bool {
             if (a.x != b.x) return a.x < b.x;
             return a.type < a.type;
         });
    int cnt = 0;
    int ans = 0;
    for(Event e : events) {
        if (e.type == 'E') {
            ++cnt;
            ans = max(cnt,ans);
        }
        else
            --cnt;
    }

    cout << ans << endl;

}
```

5 Fenwick

```
struct FenwickTree {
    vector<int> tri;
    FenwickTree(int N) : tri(N+10, 0) {}
    void add(int x, int d) {
        for (int i = x + 1; i < tri.size(); i += i&(-i)) {
            tri[i] += d;
        }
    }
    int query(int x) {
        int ans = 0;
        for (int i = x + 1; i > 0; i -= i&(-i)) {
            ans += tri[i];
        }
        return ans;
    }
    void pr() {
        for(int i = 0; i < (int)tri.size(); i++)
            cout << i+1 << ' ';
        cout << endl;
        for(int i = 0; i < (int)tri.size(); i++)
            cout << tri[i] << ' ';
        cout << endl;
    }
};
```

6 KMP

```
struct KMP
{
        string needle;
```

```
	vector<int> T;
	KMP(const string needle)
	{
		this->needle = needle;
		T = vector<int>(needle.size() + 1);
		int i = 0 , j = -1;
		T[0] = -1;
		while(i < needle.size())
		{
			while(j >= 0 && needle[i] != needle[j])j = T[j];
			T[++i] = ++j;
		}
	}
	vector<int> match(const string hay)
	{
		vector<int> V;
		int i = 0 , j = 0;
		while(i < hay.size())
		{
			while(j >= 0 && hay[i] != needle[j])j = T[j];
			++i;++j;
			if(j == needle.size())
			{
				V.push_back(i - j);
				j = T[j];
			}
		}
		return V;
	}
};
```

7 MST

```
class DisjointSet {
  int N;
  int ncomp;
  vector<int> par;
  vector<int> rank;

public:
  DisjointSet(size_t _N) : N(_N), ncomp(_N), par(_N, -1), rank(_N, 0) {}
  void reset() {
    par.assign(N, -1);
    rank.assign(N, 0);
    ncomp = N;
  }
  int size() const {
    return ncomp;
  }
  int find_rep(int u) {
    return par[u] < 0 ? u : par[u] = find_rep(par[u]);
  }
  bool union_rep(int u, int v) {
    int u_root = find_rep(u);
    int v_root = find_rep(v);
    if (u_root == v_root)
      return false;
    if (rank[u_root] > rank[v_root])
      par[v_root] = u_root;
    else {
      par[u_root] = v_root;
      if (rank[u_root] == rank[v_root])
        rank[v_root] = rank[u_root] + 1;
    }
    --ncomp;
    return true;
  }
};

struct Edge {
  int u, v;
  int cost;
  Edge(int _u, int _v, int _cost) : u(_u), v(_v), cost(_cost) {}
};

class CostCmp {
public:
  bool operator()(const Edge& e1, const Edge& e2) {
    if (e1.cost != e2.cost) return e1.cost < e2.cost;
    if (e1.u != e2.u) return e1.u < e2.u;
    return e1.v < e2.v;
  }
};

// vector<bool> in_mst;
long long kruskal(int N, vector<Edge>& Edges) {
// in_mst.assign( edges.size(), false );
```

```
   sort(Edges.begin(), Edges.end(), CostCmp());
   DisjointSet dset(N);
   long long cost = 0;
   for (int j = 0; j < int(Edges.size()) && int(dset.size()) > 1; ++j) {
      if (dset.union_rep(Edges[j].u, Edges[j].v)) {
         cost += Edges[j].cost;
      // in_mst[ edges[j].id ] = true;
      }
   }
   return cost;
}

int main(int argc, char* argv[]) {
   int N, M;
   vector<Edge> edges;
   scanf("%d %d", &N, &M);
   for (int j = 0; j < M; ++j) {
      int u, v, cost;
      scanf("%d %d %d", &u, &v, &cost);
      edges.push_back(Edge(u, v, cost));
   }

   long long res = kruskal(N, edges);
   printf("%lld\n", res);

   return 0;
}
```

8 SQRTDecomposition

```
struct RangeSumQuery {
    int NC,NR;
    vector<vector<int> > B;
    vector<int> ROWSUM;
    RangeSumQuery(const vector<int>& V) {
        NC = sqrt(V.size());
        NR = V.size() / NC + 1;
        B = vector<vector<int> >(NR, vector<int>(NC));
        ROWSUM = vector<int>(NR, 0);
        for(int i = 0; i < int(V.size()); i++) {
            int row = i / NC;
            int col = i % NC;
            B[row][col] = V[i];
            ROWSUM[row] += V[i];
        }

        for(int r = 0; r < NR; ++r) {
            for(int c = 0; c < NC; ++c) {
                cout << B[r][c] << ' ';
            }
            cout << "SUM : " << ROWSUM[r] << endl;
        }
    }

    void update(int pos, int delta) {
        int row = pos / NC;
        int col = pos % NC;
        B[row][col] += delta;
        ROWSUM[row] += delta;
    }

    int query(int a, int b) {
        int rowa = a / NC;
        int cola = a % NC;

        int rowb = b / NC;
        int colb = b % NC;

        int sum = 0;

        if (rowa == rowb) {
            for(int j = cola; j <= colb; j++)
                sum += B[rowa][j];
            return sum;
        }

        for(int j = cola; j < NC; j++) {
            sum += B[rowa][j];
        }

        for(int r = rowa+1; r < rowb; r++) {
            sum += ROWSUM[r];
        }

        for(int j = 0; j <= colb; j++) {
            sum += B[rowb][j];
        }
```

```
        return sum;
    }
};
```

9 SegmentTree

```
struct SegmentNode{
        int sz = 1;
        bool HasCarry = 0;
        void join(const SegmentNode &l, const SegmentNode &r){
                sz = l.sz + r.sz;
        }
        void update(){ HasCarry = 1; }
        void clear(){ HasCarry = 0; }
};
template<class T>
struct SegmentTree
{
        vector<T> V;
        int N;
        SegmentTree(int N) : V(4*N), N(N) {}
        void create(const vector<typename T::Init> &VEC,int n = 1,int b =
            0,int e = -1)
        {
                if (e == -1) e = N - 1;
                if (b == e) V[n] = T(VEC[b]);
                else {
                        create(VEC,2*n,b,(e+b)/2);
                        create(VEC,2*n+1,(e+b)/2+1,e);
                        V[n] = V[2*n] + V[2*n+1];
                }
        }
        T query(int i,int j, int n = 1,int b = 0,int e = -1)
        {
                if (e == -1)e = N - 1;
                if (i <= b && e <= j) return V[n];
                else {
                        if(V[n].HasCarry) {
                                V[2*n ].update(V[n].carry);
                                V[2*n+1].update(V[n].carry);
                                V[n].clear();
                        }
                        int mid = (b+e)/2;
                        if(i > mid)return query(i,j,2*n+1,mid+1,e);
                        if(j <=mid)return query(i,j,2*n,b,mid);
                        return query(i,j,2*n,b,mid) +
                            query(i,j,2*n+1,mid+1,e);
                }
        }
        void update(int i,int j,int v,int n = 1,int b=0,int e=-1)
        {
                if (e == -1) e = N - 1;
                if (i <= b && e <= j) V[n].update(v);
                else if (i > e || j < b) return;
                else {
                        if(V[n].HasCarry) {
                                V[2*n ].update(V[n].carry);
                                V[2*n+1].update(V[n].carry);
                                V[n].clear();
                        }
                        int mid = (b+e)/2;
                        update(i,j,v,2*n,b,mid);
                        update(i,j,v,2*n+1,mid+1,e);
                        V[n] = V[2*n] + V[2*n+1];
                }
        }
        int findLastIndex( bool (*isOk)(T) ){ // almost never needed
                int n = 1;
                T acum;
                acum.sz = 0;
                while(V[n].sz > 1){
                        int l = 2*n , r = 2*n+1;
                        T newAcum = (acum.sz == 0 ? V[r] : (V[r] + acum));
                        if(isOk(newAcum)){
                                n = r;
                        }else{
                                acum = newAcum;
                                n = l;
                        }
                }
                return N - acum.sz - 1;
        }
};
```