ICPC Cheat Sheet

2024-09-12 By Micheal

Out of template

Compile Command

```
g++ a.cpp -Wall -std=gnu++20 -O2 -o a
```

RD

cin

include

Math

Convolution

```
const unsigned long long Mod(998244353);
unsigned W[21], IW[21];
inline void Init() {
  IW[20] = Pow(W[20] = Pow(3, 952), Mod - 2);
  for (unsigned i(20); i; --i)
    W[i - 1] = (unsigned long long)W[i] * W[i] % Mod;
  for (unsigned i(20); i; --i)
    IW[i - 1] = (unsigned long long)IW[i] * IW[i] % Mod;
}
inline void DIT(unsigned *f, unsigned N) {
  for (unsigned i(1), I(1); !(i >> N); i <<= 1, ++I) {
    unsigned long long w(W[I]), Cur(1);
    for (unsigned j(0); !(j >> N); ++j, Cur = Cur * w % Mod)
      if (!(j & i)) {
        unsigned long long TmpA(f[j]), TmpB(f[j ^ i] * Cur % Mod);
        Mn(f[j] = TmpA + TmpB);
        Mn(f[j ^ i] = (Mod + TmpA - TmpB));
      }
  }
}
inline void DIF(unsigned *f, unsigned N) {
  for (unsigned i(1 << (N - 1)), I(N); i; i >>= 1, --I) {
    unsigned long long w(IW[I]), Cur(1);
    for (unsigned j(0); !(j >> N); ++j, Cur = Cur * w % Mod)
      if (!(j & i)) {
        unsigned long long TmpA(f[j]), TmpB(f[j ^ i]);
        Mn(f[j] = TmpA + TmpB);
        f[j ^ i] = (Mod + TmpA - TmpB) * Cur % Mod;
      }
  }
}
inline void Mul(unsigned *A, unsigned *B, unsigned Ln, unsigned Rn) {
  unsigned Len(Ln + Rn - 1), N(0);
  while ((1 << N) < Len) ++N;
  unsigned long long InvN(Pow(1 << N, Mod - 2));
  DIF(A, N), DIF(B, N);
  for (unsigned i((1 << N) - 1); ~i; --i)
    Tmp[i] = (unsigned long long)A[i] * B[i] % Mod;
  DIT(Tmp, N);
  for (unsigned i(0); i < Len; ++i) A[i] = A[i] * InvN % Mod;
}
```

ExGCD

```
long long Exgcd(long long x, long long y, long long &X, long long &Y) {
  if(y) {
    long long ExTmp(Exgcd(y, x % y, Y, X));
    Y -= X * (x/y);
    return tmp;
  }
  X = 1;
  Y = 0;
  return x;
}
```

Calculate Number Theory Inverse in Linear Time

```
Inv[1] = 1;
Inv[i] = (Mod - Mod / i) * Inv[Mod % i] % Mod;
```

String

Suffix Array

```
unsigned n, t, A, B;
unsigned SA[2000005], RK[2000005], BucSize;
unsigned Tmp[(n + 1) << 1], Bucket[max((unsigned)256, n + 1)], Cnt(0), Cons(1);
char a[2000005];

memset(RK, 0, (n + 1) << 3);
BucSize = 255;
memset(Bucket, 0, (BucSize + 1) << 2);
for (unsigned i(1); i <= n; ++i) ++Bucket[RK[i] = x[i]];
for (unsigned i(1); i <= BucSize; ++i) Bucket[i] += Bucket[i - 1];
for (unsigned i(1); i <= n; ++i) SA[Bucket[RK[i]]--] = i;
while (Cnt < n) {
  memset(Bucket, 0, (BucSize + 1) << 2);
  for (unsigned i(1); i <= n; ++i) ++Bucket[RK[i]];
  for (unsigned i(1); i <= BucSize; ++i) Bucket[i] += Bucket[i - 1];
  unsigned Top(0);
  for (unsigned i(n); i; --i) if(SA[i] > Cons) Tmp[++Top] = SA[i] - Cons;
  for (unsigned i(n - Cons + 1); i <= n; ++i) Tmp[++Top] = i;
  for (unsigned i(1); i <= n; ++i) SA[Bucket[RK[Tmp[i]]]--] = Tmp[i];
  memcpy(Tmp, RK, (n + 1) << 3);
  RK[SA[1]] = 1, Cnt = 1;
  for (unsigned i(2); i <= n; ++i)
    if((Tmp[SA[i]] ^ Tmp[SA[i - 1]]) || (Tmp[SA[i] + Cons] ^ Tmp[SA[i - 1] + Cons]))
      RK[SA[i]] = ++Cnt;
    else RK[SA[i]] = Cnt;
  Cons <<= 1, BucSize = Cnt;
}
```

Suffix Automaton

```
unsigned m, n, Len;
char SPool[1100005], *S(SPool);
struct Node{
  Node *E[26], *Fail;
  unsigned Len;
  inline Node*Add(char c);
}N[2200005], *CntN(N);
inline Node* Node::Add(char c) { // Input the last Node, Return the next Node
  if(E[c]) {
    if(E[c]->Len == Len + 1) return E[c];
    Node* Copy(++CntN), *Back(this), *Ori(E[c]);
    *Copy = *E[c], Copy->Len = Len + 1, Ori->Fail = Copy;
    while (Back && (Back->E[c] == Ori)) Back->E[c] = Copy, Back = Back->Fail;
    return Copy;
  }
  Node*Cur(++CntN), *Back(this);
  Cur->Len = Len + 1;
  while (Back && (!Back->E[c])) Back->E[c] = Cur, Back = Back->Fail;
  if(!Back) {Cur->Fail = N; return Cur;}
  if(Back->E[c]->Len == Back->Len + 1) {Cur->Fail = Back->E[c]; return Cur;}
  Node*Copy(++CntN), *Ori(Back->E[c]);
  *Copy = *Ori, Copy->Len = Back->Len + 1;
  Cur->Fail = Ori->Fail = Copy;
  while (Back && (Back->E[c] == Ori)) Back->E[c] = Copy, Back = Back->Fail;
  return Cur;
}
signed main() {
  n = RD(), m = RD();
  for (unsigned i(1); i <= m; ++i) {
    scanf("%s", S), Len = strlen(S);
    Node* Cur(N);
    for (unsigned j(0); j < Len; ++j) Cur = Cur->Add(S[j] - 'a');
    S = S + Len;
  }
  return Wild_Donkey;
}
```

Graph

Dinic

```
int a[1005][1005];
unsigned c[205];
unsigned char b[1005][1005];
int C;
unsigned m, n, P;
unsigned Cnt(0), Ans(0), Tmp(0);
struct Node;
struct Edge {
  Node* To;
  unsigned Inv, Con;
};
struct Node {
  vector<Edge> E;
  unsigned Frm, Dep;
}N[205];
inline void Link (Node* x, Node* y, unsigned Val) {
  x->E.push_back({y, y->E.size(), Val});
  y->E.push_back({x, x->E.size() - 1, 0});
}
inline char BFS() {
  Node* Que[P + 2], **Hd(Que), **Tl(Que);
  for (Node* i(N + P + 1); i >= N; --i) i->Frm = 0, i->Dep = 0x3f3f3f3f;
  (*(++Hd) = N)->Dep = 0;
  while (Tl != Hd) {
    Node* Cur(*(++Tl));
    for (auto i:Cur->E) if((i.Con) && (i.To->Dep >= 0x3f3f3f3f))
      (*(++Hd) = i.To)->Dep = Cur->Dep + 1;
  }
  return N[P + 1].Dep < 0x3f3f3f3f;
}
inline unsigned DFS(Node* x, unsigned Come) {
  if(x == N + P + 1) return Come;
  unsigned Gone(0);
  for (unsigned &i(x->Frm); Come && (i < x->E.size()); ++i)
    if (x->E[i].Con && (x->E[i].To->Dep > x->Dep)) {
      unsigned Succ(DFS(x->E[i].To, min(Come, x->E[i].Con)));
      Come -= Succ, x->E[i].Con -= Succ;
      x->E[i].To->E[x->E[i].Inv].Con += Succ, Gone += Succ;
    }
  return Gone;
}

Link(N, N + i, C); // Add Edges
while (BFS()) Tmp += DFS(N, 0x3f3f3f3f);
//Tmp is Answer
```

HLPP

```
unsigned Hd(0), Tl(0), Gap[1205], m, n, Cnt(0), C, D, t, Tmp(0);
struct Node;
struct Edge {
  Node *To;
  Edge *Nxt;
  unsigned Contain;
}E[240005], *CntE(E - 1);
struct Node {
  Edge *Fst;
  unsigned Dep, Contain;
}N[1205], *Qu[1205], *A, *B, *S, *T;
struct Que {
  Node *P;
  inline const char operator<(const Que &x) const {
    return this->P->Dep < x.P->Dep;
  }
};
priority_queue <Que> Q;
signed main() {
  n = RD(), m = RD(), S = N + RD(), T = N + RD();
  for (register unsigned i(1); i <= m; ++i) {
    A = N + RD(), B = N + RD(), C = RD();
    if(A == B) continue;
    (++CntE)->Nxt = A->Fst;
    A->Fst = CntE;
    CntE->To = B;
    CntE->Contain = C;
    (++CntE)->Nxt = B->Fst;
    B->Fst = CntE;
    CntE->To = A;
  }
  T->Dep = 1, Qu[++Tl] = T;
  register Node *x;
  while(Hd < Tl) {
    x = Qu[++Hd];
    register Edge *Sid(x->Fst);
    while (Sid) {
      if((!(Sid->To->Dep)) && (!(Sid->Contain))) {
        ++Gap[Sid->To->Dep = x->Dep + 1];
        Qu[++Tl] = Sid->To;
      }
      Sid = Sid->Nxt;
    }
  }
  --Gap[S->Dep];
  ++Gap[S->Dep = n + 1];
  register Que Pu;
  register Edge *Sid(S->Fst);
  while (Sid) {
    if(Sid->Contain) {
      if(Sid->To != T && (!(Sid->To->Contain))) {
        Pu.P = Sid->To;
        Q.push(Pu);
      }
      Sid->To->Contain += Sid->Contain;
      (Sid + 1)->Contain = Sid->Contain;
      Sid->Contain = 0;
    }
    Sid = Sid->Nxt;
  }
  while(Q.size()) {
    x = (Q.top()).P, Q.pop();
    register unsigned Real;
    Sid = x->Fst;
    Tmp = 0x3f3f3f3f;
    while(Sid) {
      if(Sid->Contain) {
        if(Sid->To->Dep + 1 == x->Dep) {
          Real = min(x->Contain, Sid->Contain);
          if(!Real) {Sid = Sid->Nxt; continue;}
          x->Contain -= Real;
          Sid->Contain -= Real;
          E[(Sid - E) ^ 1].Contain += Real;
          if(Sid->To != S && Sid->To != T && (!(Sid->To->Contain))) {
            Pu.P = Sid->To, Q.push(Pu);
          }
          Sid->To->Contain += Real;
          if(!(x->Contain)) break;
        } else Tmp = min(Tmp, Sid->To->Dep);
      }
      Sid = Sid->Nxt;
    }
    if(x->Contain) {
      if(!(--Gap[x->Dep])) {
        for (register unsigned i(1); i <= n; ++i) {
          if(N + i != S && N + i != T && N[i].Dep > x->Dep) {
            N[i].Dep = n + 2;
          }
        }
      }
      ++Gap[x->Dep = Tmp + 1];
      Pu.P = x;
      Q.push(Pu);
    }
  }
  printf("%u\n", T->Contain);
  return Wild_Donkey;
}
```

Data structure

Lichao Tree

```
unsigned a[10005], l[10005], L[10005];
unsigned long long f[10005], N, D, Ans(0x3f3f3f3f3f3f3f3f);
unsigned m, n(0), C, t;
unsigned Cnt(0), Tmp(0);
struct Line { // y = Kx + B
  unsigned long long K, B;
  inline unsigned long long F(const unsigned long long y) const {return B + y * K;}
  inline const char Com (const Line &x, const unsigned long long y) const {
    return F(y) < x.F(y);
  }
}A;
struct Node {
  Node *LS, *RS;
  Line Val;
}T[100005], *CntT(T);
inline void Ins(Node* x, unsigned L, unsigned R) {//Insert Line A
  if(L == R) {if(A.Com(x->Val, L)) x->Val = A; return; }
  unsigned Mid((L + R) >> 1);
  if(A.Com(x->Val, Mid)) swap(x->Val, A);
  if(A.K > x->Val.K) {
    if(!(x->LS)) x->LS = ++CntT, x->LS->Val = x->Val, x->LS->LS = x->LS->RS = NULL;
    Ins(x->LS, L, Mid);
  } else {
    if(!x->RS) x->RS = ++CntT, x->RS->Val = x->Val, x->RS->LS = x->RS->RS = NULL;
    Ins(x->RS, Mid + 1, R);
  }
  return;
}
inline void Find(Node* x, unsigned L, unsigned R) { // Find f(C)
  D = min(D, x->Val.F(C));
  if(L == R) return;
  unsigned Mid((L + R) >> 1);
  if(C <= Mid) {if(x->LS) Find(x->LS, L, Mid);}
  else {if(x->RS) Find(x->RS, Mid + 1, R);}
}
```

ZKW Tree

下标从 1 到 n.

```
unsigned long long T[262144], Tag[262144];  //>= 2 (n + 2)
void Build() {
  for (unsigned i(N - 1); ~i; --i) T[i] = T[i << 1] + T[(i << 1) + 1];
}
void Edit(unsigned L, unsigned R, unsigned long long V) { //[L, R] += V;
  L = L - 1 + N, R = R + 1 + N;
  unsigned long long LLen(0), RLen(0);
  for (unsigned Len(1); L ^ R ^ 1; L >>= 1, R >>= 1, Len <<= 1) {
    T[L] += V * LLen, T[R] += V * RLen;
    if (!(L & 1)) Tag[L ^ 1] += V, LLen += Len;
    if (R & 1) Tag[R ^ 1] += V, RLen += Len;
  }
  while (L) T[L] += LLen * V, T[R] += RLen * V, L >>= 1, R >>= 1;
}
unsigned long long Qry(unsigned L, unsigned R) { // Qry Sum [L, R]
  L = L - 1 + N, R = R + 1 + N;
  unsigned long long Rt(0), LLen(0), RLen(0);
  for (unsigned Len(1); L ^ R ^ 1; L >>= 1, R >>= 1, Len <<= 1) {
    Rt += Tag[L] * LLen, Rt += Tag[R] * RLen;
    if (!(L & 1)) Rt += T[L ^ 1] + Tag[L ^ 1] * Len, LLen += Len;
    if (R & 1) Rt += T[R ^ 1] + Tag[R ^ 1] * Len, RLen += Len;
  }
  while (L) Rt += Tag[L] * LLen, Rt += Tag[R] * RLen, L >>= 1, R >>= 1;
  return Rt;
}
signed main() {
  n = RD(), N = 1;
  while (N < n + 2) N <<= 1;
  memset(T + N, N << 3, 0), memset(Tag, N << 4, 0);
  for (unsigned i(1); i <= n; ++i) T[N + i] = RD();
  Build();
}
```

Link Cut Tree

- 0 Query: 查询 B, C 路径异或和, 保证联通
- 1 Link: 若 B, C 不连通, 则加边 B-C

- 2 Cut: 若存在 B-C 边, 断之
- 3 Change: 将 B 的权值修改为 C

Link: 若 B, C 未

```
unsigned n, m;
unsigned A, B, C;
void *Stack[100005];
struct Node {
  Node *Son[2], *Fa;
  char Tag;
  unsigned Value, Sum;
  inline char RealFather() {
    return Fa && (Fa->Son[0] == this || Fa->Son[1] == this);
  }
  inline char Side() { return Fa->Son[1] == this; }
  void Update() {
    Sum = Value;
    if (Son[0]) Sum ^= Son[0]->Sum;
    if (Son[1]) Sum ^= Son[1]->Sum;
    return;
  }
  void Push_Down() {
    if (Tag) {
      Tag = 0, swap(Son[0], Son[1]);
      if (Son[0]) Son[0]->Tag ^= 1;
      if (Son[1]) Son[1]->Tag ^= 1;
    }
  }
  void Rotate() {
    Node *Father(Fa);
    char xSide(Side());
    if ((Fa = Father->Fa) && Father->RealFather()) Fa->Son[Father->Side()] = this;
    Father->Fa = this;
    if (Father->Son[xSide] = Son[xSide ^ 1]) Father->Son[xSide]->Fa = Father;
    Son[xSide ^ 1] = Father;
    Father->Update(), Update();
  }
  void Splay() {
    unsigned Head(0);
    Node *Cur(this);
    while (Cur->RealFather()) Stack[++Head] = Cur, Cur = Cur->Fa;
    Cur->Push_Down();
    if (!Head) return;
    for (unsigned i(Head); i; --i) ((Node *)Stack[i])->Push_Down();
    Cur = this;
    while (Cur->RealFather()) {
      if (Cur->Fa->RealFather())
```

```
        ((Cur->Side() ^ Cur->Fa->Side()) ? Cur : Cur->Fa)->Rotate();
      Cur->Rotate();
    }
  }
  void Access() {
    // printf("Access %u\n", this);
    Splay(), Son[1] = NULL, Update();  // Delete x's right son
    Node *Cur(this), *Father(Fa);
    while (Father) {
      Father->Splay(), Father->Son[1] = Cur;          // Change the right son
      Cur = Father, Father = Cur->Fa, Cur->Update();  // Go up
    }
    return Splay();
  }
  Node *Find_Root() {  // Find the root
    Access(), Push_Down();
    Node *Cur(this);
    while (Cur->Son[0]) Cur = Cur->Son[0], Cur->Push_Down();
    return Cur;
  }
} N[100005];
signed main() {
  n = RD(), m = RD();
  for (unsigned i(1); i <= n; ++i) N[i].Value = N[i].Sum = RD();
  for (unsigned i(1); i <= m; ++i) {
    A = RD(), B = RD(), C = RD();
    switch (A) {
      case 0: {                        // Query
        N[B].Access(), N[B].Tag ^= 1;  // Makeroot(B)
        N[C].Access();
        printf("%u\n", N[C].Sum);
        break;
      }
      case 1: {                        // Link
        N[B].Access(), N[B].Tag ^= 1;  // Makeroot(B)
        if (N[C].Find_Root() != N + B) N[B].Fa = N + C;
        break;
      }
      case 2: {                        // Cut
        N[B].Access(), N[B].Tag ^= 1;  // Makeroot(B)
        if (N[C].Find_Root() == N + B) {
          if (N[B].Fa == N + C && !(N[B].Son[1]))
            N[B].Fa = N[C].Son[0] = NULL, N[C].Update();
        }
        break;
      }
      case 3: {  // Change
        N[B].Splay(), N[B].Value = C, N[B].Update();
        break;
      }
    }
  }
  return Wild_Donkey;
}
```