

1 Data Structure

1.1 Segment Tree

```

1 struct SegT{
2     const int MAXN = 1e5+5;
3     int tr[MAXN*4], arr[MAXN], tag[MAXN*4];
4
5     int combine(int a, int b){
6         return max(a,b);
7     }
8
9     void build(int idx, int l, int r){
10        if(l==r){
11            tr[idx] = arr[l];
12        }else{
13            int m = (l+r)/2;
14            build(idx*2,l,m);
15            build(idx*2+1,m+1,r);
16            tr[idx] = combine(tr[idx*2],tr[idx
17                *2+1]);
18        }
19
20        void push(int idx){
21            if(tag[idx]){
22                tr[idx<<1] = max(tr[idx<<1], tag[
23                    idx]);
24                tr[idx<<1|1] = max(tr[idx<<1|1],
25                    tag[idx]);
26                tag[idx<<1] = max(tag[idx<<1], tag
27                    [idx]);
28                tag[idx<<1|1] = max(tag[idx<<1|1],
29                    tag[idx]);
30                tag[idx] = 0;
31            }
32
33            void modify(int ql, int qr, int val, int
34                idx, int l, int r){
35                if(l!=r) push(idx); //當節點並非葉節點
36                    時·下推標記
37                if(ql <= l && r <= qr){
38                    tr[idx] = max(tr[idx],val);
39                    tag[idx] = max(tag[idx],val);
40                    return;
41                }
42                int m = (l+r)/2;
43                if(qr > m) modify(ql, qr, val, idx
44                    *2+1, m+1, r);
45                if(ql <= m) modify(ql, qr, val, idx*2,
46                    l, m);
47                tr[idx] = combine(tr[idx<<1],tr[idx
48                    <<1|1]);
49            }
50
51            int query(int ql, int qr, int idx, int l,
52                int r){
53                if(l!=r) push(idx);
54                if(ql <= l && r <= qr){
55                    return tr[idx];
56                }
57                int m = (l+r)/2;

```

```

49         if(ql > m){
50             return query(ql, qr, idx*2+1, m+1,
51                 r);
52         }
53         if(qr <= m){
54             return query(ql, qr, idx*2, l, m);
55         }
56         return combine(query(ql, qr, idx*2, l,
57             m), query(ql, qr, idx*2+1, m+1, r
58             ));
59     }
60 }

```

1.2 Treap

```

1 struct Treap{
2     Treap *l, *r;
3     int val, size, sum;
4     Treap(int v): l(nullptr), r(nullptr), val(
5         v), size(1), sum(v){}
6     void pull();
7
8     void Treap::pull(){
9         size = 1, sum = val;
10        if(l!=nullptr) size += l->size, sum += l->
11            sum;
12        if(r!=nullptr) size += r->size, sum += r->
13            sum;
14    }
15
16    int sz(Treap *t){
17        return (t==nullptr ? 0 : t->size);
18    }
19
20    Treap *merge(Treap *a, Treap *b){
21        if(a==nullptr) return b;
22        if(b==nullptr) return a;
23        if(rand()%(a->size+b->size) < a->size){
24            a->r = merge(a->r,b);
25            a->pull();
26            return a;
27        }else{
28            b->l = merge(a,b->l);
29            b->pull();
30            return b;
31        }
32    }
33
34    void split(Treap *t, Treap *&a, Treap *&b,
35        int k){
36        if(t==nullptr){
37            a = b = nullptr;
38            return;
39        }
40        if(sz(t->l) < k){
41            a = t;
42            split(t->r,a->r,b,k-sz(t->l)-1);
43            a->pull();
44        }else{
45            b = t;
46            split(t->l,a,b->l,k);
47            b->pull();
48        }
49    }

```

2 Graphs

2.1 dijkstra

```

1 priority_queue<pair<int,int>,vector<pair<int
2     ,int>>,greater<pair<int,int>>> pq;
3 pq.push({0,s});
4 dis[s] = 0;
5 inq[s] = 1;
6 while(!pq.empty()){
7     auto [w,u] = pq.top(); pq.pop();
8     inq[u] = 0;
9     for(auto [v,w] : adj[u]){
10        if(dis[v] > dis[u]+ w){
11            dis[v] = dis[u]+w;
12            if(!inq[v]){
13                pq.push({dis[v],v});
14                inq[v] = 1;
15            }
16        }
17    }

```

3 Number Theory

3.1 FFT

```

1 typedef complex<double> cp;
2
3 const double pi = acos(-1);
4 const int NN = 131072;
5
6 struct FastFourierTransform{
7     /*
8     Iterative Fast Fourier Transform
9
10    How this works? Look at this
11
12    0th recursion 0(000) 1(001) 2(010)
13                  3(011) 4(100) 5(101) 6(110)
14                  7(111)
15
16    1th recursion 0(000) 2(010) 4(100)
17                  6(110) | 1(011) 3(011) 5(101)
18                  7(111)
19
20    2th recursion 0(000) 4(100) | 2(010)
21                  6(110) | 1(011) 5(101) | 3(011)
22                  7(111)
23
24    3th recursion 0(000) | 4(100) | 2(010) |
25                  6(110) | 1(011) | 5(101) | 3(011)
26                  7(111)

```

All the bits are reversed => We can save the reverse of the numbers in an array!

```

18 */
19 int n, rev[NN];
20 cp omega[NN], iomega[NN];
21 void init(int n_){
22     n = n_;
23     for(int i = 0; i < n; i++){
24         //Calculate the nth roots of unity
25         omega[i] = cp(cos(2*pi*i/n), sin(2*pi*
26             i/n));
27         iomega[i] = conj(omega[i]);
28     }
29     int k = __lg(n);
30     for(int i = 0; i < n; i++){
31         int t = 0;
32         for(int j = 0; j < k; j++){
33             if(i & (1<<j)) t |= (1<<(k-j-1));
34         }
35         rev[i] = t;
36     }
37 }
38
39 void transform(vector<cp> &a, cp* xomega){
40     for(int i = 0; i < n; i++){
41         if(i < rev[i]) swap(a[i],a[rev[i]]);
42         for(int len = 2; len <= n; len <= 1){
43             int mid = len >> 1;
44             int r = n/len;
45             for(int j = 0; j < n; j += len){
46                 for(int i = 0; i < mid; i++){
47                     cp tmp = xomega[r*i] * a[j+mid+i];
48                     a[j+mid+i] = a[j+i] - tmp;
49                     a[j+i] = a[j+i] + tmp;
50                 }
51             }
52         }
53     }
54 }
55
56 void fft(vector<cp> &a){ transform(a,omega
57     ); }
58
59 void ifft(vector<cp> &a){ transform(a,
60     iomega); for(int i = 0; i < n; i++) a[i]
61     /= n; }
62 } FFT;

```

3.2 NTT

```

1 const int N = 1e5+5, MOD = 998244353, G = 3;
2
3 int fastpow(int n, int p){
4     int res = 1;
5     while(p){
6         if(p&1) res = res * n % MOD;
7         n = n * n % MOD;
8         p >>= 1;
9     }
10    return res;
11 }
12
13 struct NTT{
14     int n, inv, rev[N];
15     int omega[N], iomega[N];

```

```

16 void init(int n_){
17     n = 1;
18     while(n < n_) n<<=1;
19     inv = fastpow(n,MOD-2);
20     int k = __lg(n);
21     int x = fastpow(G, (MOD-1)/n);
22     omega[0] = 1;
23     for(int i = 1; i < n; i++){
24         omega[i] = omega[i-1] * x % MOD;
25         iomega[n-1] = fastpow(omega[n-1],MOD
26             -2);
27         for(int i = n-2; i >= 0; i--){
28             iomega[i] = iomega[i+1] * x %
29                 MOD;
30         }
31         rev[i] = t;
32     }
33 }
34 void transform(int *a, int *xomega){
35     for(int i = 0; i < n; i++){
36         if(i < rev[i]) swap(a[i],a[rev[i]
37             ]]);
38         for(int len = 2; len <= n; len <= 1){
39             int mid = len>>1;
40             int r = n/len;
41             for(int j = 0; j < n; j += len){
42                 for(int i = 0; i < mid; i++){
43                     int tmp = xomega[r*i] *
44                         a[j+mid+i] % MOD;
45                     a[j+mid+i] = (a[j+i] -
46                         tmp + MOD) % MOD;
47                     a[j+i] = (a[j+i]+tmp)%
48                         MOD;
49                 }
50             }
51         }
52     }
53     void dft(int *a){transform(a,omega);}
54     void idft(int *a){transform(a,iomega);}
55     for(int i = 0; i < n; i++) a[i] = a[i]
56         *inv %MOD;
57 }
58 int tmp[8][N];
59 void copy_(int *a, int *b, int m){
60     for(int i = 0; i < m; i++){
61         a[i] = b[i];
62     }
63     for(int i = m; i < n; i++){
64         a[i] = 0;
65     }
66 }
67 void copy(int *a, int *b, int m){
68     for(int i = 0; i < m; i++){
69         a[i] = b[i];
70     }
71 }
72 //B_{k+1} = B_k(2-AB_k) (mod MOD)
73 void inverse(int *a, int *b, int m){
74     //Uses tmp[0], tmp[1]
75     if(m==1){
76         b[0] = fastpow(a[0],MOD-2);
77         return;
78     }
79     inverse(a,b,m>>1);
80     init(m<<1);
81     copy_(tmp[0],a,m); copy_(tmp[1],b,m
82         >>1);
83     dft(tmp[0]); dft(tmp[1]);
84     for(int i = 0; i < n; i++){ tmp[0][i] =
85         tmp[1][i]*(2-tmp[0][i]*tmp[1][i]
86             %MOD+MOD)%MOD;
87     }
88     idft(tmp[0]);
89     copy(b,tmp[0],m);
90 }
91 //Q_{k+1} = pow(2,MOD-2)(Q_k + P*pow(Q_k
92     ,MOD-2)) (mod MOD)
93 void sqrt(int *a, int *b, int m){
94     //Uses tmp[2], tmp[3]
95     if(m==1){
96         b[0] = 1;
97         return;
98     }
99     sqrt(a,b,m>>1);
100     for(int i = m; i < m<<1; i++){
101         b[i] = 0;
102     }
103     inverse(b,tmp[2],m);
104     init(m<<1);
105     for(int i = m; i < m<<1; i++){
106         b[i] = tmp[2][i] = 0;
107     }
108     int inv2 = fastpow(2,MOD-2);
109     copy_(tmp[3],a,m);
110     dft(tmp[3]); dft(tmp[2]);
111     for(int i = 0; i < n; i++){
112         tmp[3][i] = tmp[3][i]*tmp[2][i]%
113             MOD;
114     }
115     idft(tmp[3]);
116     for(int i = 0; i < m; i++){
117         b[i] = (b[i]+tmp[3][i])%MOD*inv2
118             %MOD;
119     }
120 }
121 void derivative(int *a, int *b, int m){
122     for(int i = 1; i < m; i++){ b[i-1] = a[
123         i]*i%MOD;
124     }
125     b[m-1] = 0;
126 }
127 void integral(int *a, int *b, int m){
128     for(int i = m-1; i--){ b[i] = a[i]
129         -1]*fastpow(i,MOD-2)%MOD;
130     }
131     b[0] = 0;
132 }
133 void ln(int *a, int *b, int m){
134     //Uses tmp[4], tmp[5]
135     inverse(a,b,m);
136     derivative(a,tmp[5],m);
137     init(m<<1);
138     copy_(tmp[4],b,m), copy_(tmp[5],tmp
139         [5],m);
140     dft(tmp[4]); dft(tmp[5]);
141     for(int i = 0; i < m<<1; i++){ tmp[4][i]
142         = tmp[4][i]*tmp[5][i]%MOD;
143     }
144     idft(tmp[4]);
145     integral(tmp[4],b,m);
146 }
147 void exp(int *a, int *b, int m){
148     //Uses tmp[6], tmp[7]
149     b[0] = 1;
150     for(int i = 4; j = 2; j <= m; j = i, i
151         <<=1){
152         ln(b,tmp[6],j);
153         tmp[6][0] = (a[0]+1-tmp[6][0]+
154             MOD)%MOD;
155         for(int k = 1; k < j; k++){ tmp
156             [6][k] = (a[k]-tmp[6][k]+MOD
157                 )%MOD;
158         }
159         fill(tmp[6]+j,tmp[6]+i,0);
160         dft(b), dft(tmp[6]);
161         for(int k = 0; k < i; k++){ b[k] =
162             b[k]*tmp[6][k]%MOD;
163         }
164         idft(b);
165         fill(b+j,b+i,0);
166     }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

KEEP ON THE
HARD WORK!

Contents

1 Data Structure

1

2 Graphs

3 Number Theory

1

1.1 Segment Tree 1
1.2 Treap 1
2.1 dijkstra 1

3.1 FFT 1
3.2 NTT 1