**Rooted Tree**

#include <stdio.h>

#include <stdlib.h>

#define V 13

int p[V + 1];

int rank[V + 1];

void makeSet(int x)

{

  p[x] = x;

  rank[x] = 0;

}

int findSet(int x)

{

  if (x != p[x])

  {

    p[x] = findSet(p[x]);

  }

  return p[x];

}

void link(int x, int y)

{

  if (rank[x] > rank[y])

  {

    p[y] = x;

  }

  else

  {

    p[x] = y;

    if (rank[x] == rank[y])

    {

      rank[y]++;

    }

  }

}

void unionSets(int x, int y)

{

  link(findSet(x), findSet(y));

}

int main()

{

  int i, j;

  FILE \*file = fopen("graph.txt", "r");

  if (file == NULL)

  {

    perror("Error opening file");

    return 1;

  }

  for (i = 1; i <= V; i++)

  {

    makeSet(i);

  }

  int u, v;

  while (fscanf(file, "%d %d", &u, &v) == 2)

  {

    unionSets(u, v);

  }

  fclose(file);

  int components[V + 1] = {0};

  for (i = 1; i <= V; i++)

  {

    components[findSet(i)]++;

  }

  printf("Connected Components:\n");

  int k = 1;

  for (i = 1; i <= V; i++)

  {

    if (components[i] > 0)

    {

      printf("Component %d: ", k++);

      for (j = 1; j <= V; j++)

      {

        if (findSet(j) == i)

        {

          printf("%d ", j);

        }

      }

      printf("root: %d", i);

      printf("\n");

    }

  }

  return 0;

}

**Linked List**

#include <stdio.h>

#include <stdlib.h>

typedef struct head

{

  struct object \*head;

  struct object \*tail;

  int size;

} head;

typedef struct object

{

  int data;

  struct object \*next;

  head \*prev;

} object;

head \*makeSet(int data)

{

  head \*Header = (head \*)malloc(sizeof(head));

  Header->head = (object \*)malloc(sizeof(object));

  Header->tail = (object \*)malloc(sizeof(object));

  Header->size = 1;

  Header->head->data = data;

  Header->head->next = NULL;

  Header->head->prev = Header;

  Header->tail = Header->head;

  return Header;

}

head \*findSet(head \*x)

{

  return x->head->prev;

}

int SameSet(head \*x, head \*y)

{

  return (x->head->prev == y->head->prev);

}

void Union(head \*x, head \*y)

{

  if (x->head->prev == y->head->prev)

  {

    return;

  }

  head \*X = x->head->prev;

  head \*Y = y->head->prev;

  if (X->size > Y->size)

  {

    object \*temp = Y->head, \*prev;

    while (temp)

    {

      prev = temp;

      temp->prev = X;

      temp = temp->next;

    }

    X->tail->next = Y->head;

    X->tail = prev;

    X->size += Y->size;

    return;

  }

  object \*temp = X->head, \*prev;

  while (temp)

  {

    prev = temp;

    temp->prev = Y;

    temp = temp->next;

  }

  Y->tail->next = X->head;

  Y->tail = prev;

  Y->size += X->size;

}

void Connect(head \*x, head \*y)

{

  if(!SameSet(x, y))

  {

    Union(x, y);

  }

}

void printSet(head \*x)

{

  printf("Size of the set = %d\n", x->size);

  object \*temp = x->head;

  while (temp)

  {

    printf("%d ", temp->data);

    temp = temp->next;

  }

  printf("\n");

}

int main()

{

  FILE \*file = fopen("graph.txt", "r");

  if (file == NULL)

  {

    perror("Error opening file");

    return 1;

  }

  int i;

  head \*headers[13];

  for (i = 0; i < 13; i++)

  {

    headers[i] = makeSet(i + 1);

  }

  int u, v;

  while (fscanf(file, "%d %d", &u, &v) == 2)

  {

    Connect(headers[u - 1], headers[v - 1]);

  }

  int k = 1;

  printf("The Sets are:\n");

  for (i = 0; i < 13; i++)

  {

    if (headers[i]->head->prev == headers[i])

    {

      printf("Set %d: ", k++);

      printSet(headers[i]);

      printf("root = %d\n", headers[i]->head->data);

    }

  }

  return 0;

}

**Matrix Multiplication**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

void matrixChain(int \*p, int n, int \*\*m, int \*\*s)

{

  int i, l, j, k, q;

  for (i = 1; i <= n; i++)

  {

    m[i][i] = 0;

  }

  for (l = 2; l <= n; l++)

  {

    for (i = 1; i <= n - l + 1; i++)

    {

      j = i + l - 1;

      m[i][j] = INT\_MAX;

      for (k = i; k <= j - 1; k++)

      {

        q = m[i][k] + m[k + 1][j] + p[i - 1] \* p[k] \* p[j];

        if (q < m[i][j])

        {

          m[i][j] = q;

          s[i][j] = k;

        }

      }

    }

  }

}

void parenPrint(int \*\*s, int i, int j)

{

  if (i == j)

  {

    printf("A%d", i);

  }

  else

  {

    printf("(");

    parenPrint(s, i, s[i][j]);

    parenPrint(s, s[i][j] + 1, j);

    printf(")");

  }

}

int main()

{

  int n, i;

  printf("Enter the number of matrices: ");

  scanf("%d", &n);

  int \*p = (int \*)malloc((n + 1) \* sizeof(int));

  printf("Enter the dimensions of the matrices: ");

  for (i = 0; i <= n; i++)

  {

    scanf("%d", &p[i]);

  }

  int \*\*m = (int \*\*)malloc((n + 1) \* sizeof(int \*));

  for (i = 0; i <= n; i++)

  {

    m[i] = (int \*)malloc((n + 1) \* sizeof(int));

  }

  int \*\*s = (int \*\*)malloc((n + 1) \* sizeof(int \*));

  for (i = 0; i <= n; i++)

  {

    s[i] = (int \*)malloc((n + 1) \* sizeof(int));

  }

  matrixChain(p, n, m, s);

  printf("Minimum number of multiplications: %d\n", m[1][n]);

  printf("Optimal parenthesization: ");

  parenPrint(s, 1, n);

  printf("\n");

  return 0;

}