Sallam

BITMASK

Built-in Functions

	Syntax	Usage
1-	__builtin_popcount(x)	Used to count the number of one's(set bits) in an integer.
2-	__builtin_parity(x)	Check the parity of a number.
3-	__builtin_clz(x)	Used to count the **leading** zeros of the integer.
4-	__builtin_ctz(x):	Used to count the **trailing** zeros of the given integer.

Tricks and hacks

	Syntax	Usage
1-	$(x >> i) \& 1 \mid (1 << i) \& x$	Check if i^{th} bit set or not
2-	$(1 << N)$	Left shift by N bit, equal to 2^N
3-	$63 - \_builtin\_clzll(x)$	Most set bit (last 1 in mask)
4-	$X \& (-x)$	Value of First set bit (to get index we will use loop)

Graph

Depth First Search

Works in $O(n + m)$ time where n is the number of vertices and m is the number of edges.

Implementation.

```
1- vector<ll> Graph[N] ;
2- bool visited[N] ;
3-
4- void dfs(ll node){
```

```
5-      visited[node] = true ;
6-      for(ll child : Graph[node]){
7-          if(!visited[child]){
8-              dfs(child);
9-          }
10-     }
11- }
```

Bipartiteness check.

```
1-  vector<ll> Graph[N] ;
2-  int color[N] ; //all -1
3-  bool cycle ;
4-  void dfs(ll node , ll parent , int color1)
5-  {
6-      color[node] = color1 ;
7-      for(ll child : Graph[node])
8-      {
9-          if(color[child] == -1)
10-         {
11-             dfs(child , node , !color1);
12-         }
13-         else if (color[node] == color[child] )
14-         {
15-             cycle = true ;
16-             return;
17-         }
18-     }
19- }
```

Check cycle in directed Graphs.

```
1-  vector<ll> Graph[N] ;
2-  int color[N] ;
3-  void cycle(int node, int p) {
4-      color[node] = 1;
5-      for (int w : Graph[node]) {
6-          if (color[w] == 1) {
7-              cout << ”cycle” ;
8-              exit(0);
9-          }
```

```
10-         if (color[w] == 0)
11-             cycle(w, node);
12-     }
13-     color[node] = 2;
14- }
```

Check Cycle.

```
1-  vector<ll> Graph[N] ;
2-  bool visited[N] ;
3-  bool cycle ;
4-  void dfs(ll node , ll parent){
5-      visited[node] = true ;
6-      for(ll child : Graph[node]){
7-          if(visited[child] && child != parent){
8-              cycle = true ;
9-              return;
10-         }
11-         else if (!visited[child]){
12-             dfs(child , node);
13-         }
14-     }
15- }
```

Topological Sorting using (DFS).

```
1-  vector<int>adj[N];
2-  vector<int>ans;
3-  int n,m;
4-  bool vis[N];
5-  void dfs(int node){
6-      vis[node]=true;
7-      for(auto child:adj[node]){
8-          if(!vis[child]){
9-           dfs(child);
10-         }
11-     }
12-     ans.push_back(node);
13- }
```

Sallam

```
14- int main()
15- {
16-     int n,m,u,v;
17-     cin>>n>>m>>u>>v;
18-      for(int i=0;i<m;i++){
19-     int u,v;cin>>u>>v;
20-     adj[u].push_back(v);
21-   }
22-   for(int i=1;i<=n;i++){
23-     if(!vis[i]){
24-         dfs(i);
25-     }
26-   }
27- }
```

Breadth First Search

Works in $O(n+m)$ time where n is the number of vertices and m is the number of edges.

Implementation.

```
1-  vector<ll> Graph[N];
2-  bool visited[N];
3-  ll distancee[N];
4-
5-  void BFS(ll v){
6-      queue<ll> q ;
7-      q.push(v);
8-      visited[v] = true;
9-      while (! q.empty()){
10-         ll node = q.front();
11-         q.pop() ;
12-         for(ll child : Graph[node]){
13-             if(!visited[child]){
14-                 visited[child] = true;
15-                 q.push(child);
16-                 distancee[child] = distancee[node] + 1
17- ;
```

```
18-                 }
19-             }
20-         }
    }
```

- Finding the shortest path in a graph run BFS to get distance and then run another BFS and check this inequality $p(a,b)$: $d[u] + 1 + d[v] = d[b]$
- Finding the shortest cycle in a directed unweighted graph s tart a breadth-first search from each vertex. As soon as we try to go from the current vertex back to the source vertex, we have found the shortest cycle having the source vertex.
- Find all the vertices on any shortest path between a given pair of vertices $v(a,b)$, first BFS from $\boldsymbol{a}$ we will use d_a, second BFS from $\boldsymbol{b}$ we will use d_b, and then check for each node this inequality $d_a[v] + d_b[v] = d_a[b]$

Topological Sorting using (BFS).

```
1-  for (int i = 0; i < m; i ++) {
2-          int u, v;
3-          cin >> u >> v;
4-          adj[u].push_back(v);
5-          degree[v] ++;
6-      }
7-  for (int i = 1; i <= n; i ++) {
8-        if (! degree[i]) {
9-            s.insert(i);
10-       }
11- }
12- int curr = 0;
13- while (! s.empty()) {
14-         curr = *s.begin();
15-         s.erase(s.begin());
16-         ans.push_back(curr);
17-         for (int child: adj[curr]) {
```

```
18-                 -- degree[child];
19-                 if (! degree[child]) {
20-                     s.insert(child);
21-                 }
22-             }
23-   }
24- for (int i = 0; i < ans.size(); i ++) {
25-       cout << ans[i] << ' ';
26- }
```

Finding bridges

Works in $O(n + m)$ time, A bridge is an edge whose removal makes the graph disconnected.

Implementation.

```
1-  vector<ll> Graph[N] ;
2-  ll tin [N] , tlow[N] ;
3-  bool visited[N];
4-  ll timer ;
5-
6-  void dfs(ll node , ll parent = - 1){
7-      visited[node] = true ;
8-      tin[node] = tlow[node] = timer ++ ;
9-      for(ll child : Graph[node]){
10-         if(child == parent){ continue;}
11-         if(visited[child]){
12-             tlow[node] = min(tlow[node] , tin[child]
13- );
14-         }
15-         else{
16-             dfs(child , node);
17-             tlow[node] = min(tlow[node] , tin[child]
18- );
19-             if(tlow[child] > tin[node]){
20-              cout << "edge (" << node << ',' << child
21- <<") is a bridge!\n";
22-             }
```

Sallam

```
23-             }
            }
        }
```

Dijkstra Algorithm

Time complexity $O(n \cdot log(n) + m)$

Implementation.

```
1-  vector<pair<int,int>> Graph[N];
2-  int parent[N];
3-  int dis[N];
4-
5-  priority_queue<pair<ll,ll>, vector<pair<ll,ll>>,greater<>> pq ;
6-      for (int i = 2; i <=n; ++i) {
7-          dis[i] = INF ;
8-      }
9-      // weight ,  node
10-     pq.push({0,1});
11-     while (!pq.empty())
12-     {
13-         int node = (pq.top()).second;
14-         pq.pop();
15-         for(pair<int,int> child : Graph[node])
16-         {
17-             if(dis[node] + child.second < dis[child.first])
18-             {
19-                 dis[child.first] = dis[node] + child.second;
20-                 parent[child.first] = node;
21-
22- pq.push(make_pair(dis[child.first],child.first));
23-             }
24-         }
25-     }
26-
```

Sallam

Dynamic Programming

Knapsack(pick or leave)

1D array.

```
1-  for(int i=0; i < n; i++){
2-          for(int j=W; j>=wt[i]; j--){
3-              dp[j] = max(dp[j] , val[i] + dp[j-wt[i]]);
4-          }
5-  }
```

2D array.

```
1-  for(int i = 0; i < N; ++i) {
2-       for(int j = 0; j <= W; ++j) {
3-          if(j + w[i] <= W){
4-            dp[i][j] = max(dp[i + 1][j + w[i]], dp[i][j] + v[i]);
5-          }
6-       }
7-  }
```

Interval DP

```
1-  for (int len = 0; len < n; ++len) {
2-      for (int left = 0; left + len < n; ++left) {
3-           ll right = left + len ;
4-           for (int i = left; i <= right ; ++i) {
5-                dp[left][right] = max(dp[left][right] ,
6-  dp[left][i] + dp[i+1][right]); //plus something
7-            }
8-      }
9-  }
```

Sallam

Build in range M (UVA Fast Food).

```
1-  for (int i = 1; i <= n; ++i) {
2-     for (int j = 2; j <= m; ++j) {
3-        for (int k = j - 1; k < i; ++k){
4-       if (dp[ k ][ j - 1 ] + dist[ k + 1 ][ i ] < dp[ i ][ j ]) {
5-             dp[ i ][ j ] = dp[ k ][ j - 1 ] + dist[ k + 1 ][ i ];
6-             path[ i ][ j ] = k;
7-                 }
8-           }
9-       }
10- }
11-
```

we split at point using j, so we can assume that we use first j res, and split at any point in range $[j, i]$

Matrix Chain

You will given the number of rows and columns of each matrix, ask to split the matrices to make minimum number of operations.

```
1-  for (int i = 1; i < n; i++)
2-          dp[i][i + 1] = mat_a[i] * mat_a[i + 1] * mat_b[i + 1];
3-  for (int j = 2; j < n; j++)
4-     for (int i = 1; i <= n - j; i++) {
5-         long long Min = INF;
6-         for (int k = 0; k < j; k++) {
7-           ll temp = dp[i][i + k] + dp[i + k + 1][i + j] +
8-  mat_a[i] * mat_b[i + k] * mat_b[i + j];
9-                  if ( temp < Min ) {
10-                     Min = temp;
11-                     pos[i][i + j] = i + k;
12-                 }
13-             }
14-             dp[i][i + j] = Min;
15-  }
```

Sallam

Bitmask

Visit mask S cites at day D

```
1-  for(int d = 1; d < n; d++) {
2-      for(int s = 0; s < (1<<k); s++) {
3-          total[s][d] = total[s][d-1];
4-          for(int x = 0; x < k; x++) {
5-              if(s&(1<<x)) {
6-                  total[s][d] =
7-                min(total[s][d],total[s^(1<<x)][d1]+price[x][d]);
8-              }
9-          }
10-     }
11- }
```

We can use first 8 bits to represent 1 occurrence, and second 8 bits to represent 2 occurrences.

Trees

```
1-  subtree[N];
2-  void dfs(int c, int p){
3-      int cnt = 0;
4-      for(int to : Graph[c]){
5-          if(to == p) continue;
6-          dfs(to, c);
7-          cnt += (1 + subTree[to]);
8-      }
9-      subTree[c] = cnt;
10- }
```

We can use $dp[node][len] = dp[node][x] * dp[node][y$

Sallam

LIS

```
1-  dp[0] = 1;
2-  for (int i = 1; i < n; i++) {
3-       dp[i] = 1;
4-       for (int j = 0; j < i; j++)
5-           if (arr[i] > arr[j] && dp[i] < dp[j] + 1)
6-                dp[i] = dp[j] + 1;
7-  }
```

Edit distance

```
1-  reverse(t.begin() , t.end());
2-      ll sz = s.size();
3-      for (int i = 0; i <= sz; ++i) {dp[i][0] = i ;}
4-      for (int i = 0; i <= sz; ++i) {dp[0][i] = i ;}
5-
6-      for (int i = 1; i <= sz; ++i) {
7-          for (int j = 1; j <= sz; ++j) {
8-              if(s[i-1] == t[j-1]){
9-                  dp[i][j] = dp[i-1][j-1];
10-             }else{
11-                 dp[i][j] = min({dp[i-1][j] , dp[i-1][j-1],
12- dp[i][j-1]}) + 1 ;
13-             }
14-         }
15-     }
16-     cout << dp[sz][sz]/2 << '\n';
```

LCS

```
1-  for (int i = 0; i <= m; i++)
2-      {
3-          for (int j = 0; j <= n; j++)
4-          {
5-              if (i == 0 || j == 0)
6-                  L[i][j] = 0;
7-
8-              else if (X[i - 1] == Y[j - 1])
9-                  L[i][j] = L[i - 1][j - 1] + 1;
10-
```

Sallam

```
11-                 else
12-                     L[i][j] = max(L[i - 1][j], L[i][j - 1]);
13-             }
14-         }
```