DATA STRUCTURE AND ALGORITHMS ADVANCED LAB

PHAM QUANG DUNG

Backtracking

EXERCISES

- Sudoku
- Traveling Salesman Problem
- Capacitated Bus Routing
- Taxi routing
- Balanced Course Teacher Assignment
- Balanced Academic Curriculum Problem
- Paper Reviewers Assignment Problem
- Vehicle Routing Problem

Sudoku

- Given a grid 9×9, the goal is to assign digits (from 1 to 9) to the cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	1	4	3	6	5	8	9	7
3	6	5	8	9	7	2	1	4
8	9	7	2	1	4	3	6	5
5	3	1	6	4	2	9	7	8
6	4	2	9	7	8	5	3	1
9	7	8	5	3	1	6	4	2

Sudoku

```
#include <stdio.h>

int x[9][9];
int o[3][3][10];
int h[9][10];
int c[9][10];
int found;
```

Sudoku

```
int check(int v, int i, int j){
    return !h[i][v] && !c[j][v] && !o[i/3][j/3][v];
}
void solution(){
    found = 1;
    for(int i = 0; i < 9; i++){
        for(int j = 0; j < 9; j++)
            printf("%d ",x[i][j]);
        printf("\n");
    }
    printf("----------------------\n");
}
```

Sudoku

```
void TRY(int i, int j){
    if(found) return ;
    for(int  v = 1; v <= 9; v ++){
        if(check(v,i,j)){
            x[i][j] = v;
            h[i][v] = 1;  c[j][v] = 1;  o[i/3][j/3][v] = 1;
            if(i == 8 && j == 8){
                solution();
            }else{
                if(j < 8) TRY(i,j+1);
                else TRY(i+1,0);
            }
            h[i][v] = 0;  c[j][v] = 0; o[i/3][j/3][v] = 0;
        }
    }
}
```

Sudoku

```
int main(){
    for(int   v = 1; v <= 9; v++){
        for(int i = 0; i <= 8; i++){
            h[i][v] = 0; c[i][v] = 0;
        }
        for(int i = 0; i <= 2; i++)
            for(int j = 0; j <= 2; j++)
                o[i][j][v] = 0;
    }
    found = 0;
    TRY(0,0);
}
```

Traveling Salesman Problem

- A person departs from point 0. He want to visit points 1, 2, …, n, once and come back to 0. Given $c(i,j)$ which is the traveling distance from point i to point j (i, j = 0,1,…,n), help that person to compute the shortest route.

Traveling Salesman Problem (TSP)

- Input:
  - Line 1 contains n ($1 \leq n \leq 15$)
  - Line i+1 (i = 1, 2,…, n+1) contains the i^{th} line of the matrix c
- Output:
  - Unique line contains the length of the shortest route

stdin	stdout
3 0 5 10 10 6 0 2 9 5 9 0 6 1 7 4 0	14

Traveling Salesman Problem

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 100

// data structures for input
int n;// number of points (1,2,...,n).
int c[MAX][MAX];
int cmin;// minimum distance

// solution representation
int x[MAX];
int appear[MAX];// appear[v] = 1 indicates that point v has been
                //  visited
int f;  // accumulate distance of the route under construction
int f_best; // shortest distance found so far
int x_best[MAX]; // record the best route
```

Traveling Salesman Problem

```
void input(){
    scanf("%d",&n);
    cmin = 1000000;
    for(int i = 0; i <= n;  i++){
        for(int j= 0; j <= n; j++){
            scanf("%d",&c[i][j]);
            if(i != j && cmin > c[i][j])
              cmin = c[i][j];
        }
    }
}
int check(int v, int k){
    if(appear[v] == 1) return 0;
    return 1;
}
```

Traveling Salesman Problem

```
void solution(){
    if(f + c[x[n]][0] < f_best){
        f_best = f + c[x[n]][0];
        for(int i = 0; i <= n; i++) x_best[i] = x[i];
    }
}
```

Traveling Salesman Problem

```
void TRY(int k){
    for(int v = 1; v <=n; v++){
        if(check(v,k)){
            x[k] = v;
            f += c[x[k-1]][x[k]];
            appear[v] = 1;
            if(k == n) solution();
            else{
                if(f + (n+1-k)*cmin < f_best)
                    TRY(k+1);
            }
            appear[v] = 0;
            f -= c[x[k-1]][x[k]];
        }
    }
}
```

Traveling Salesman Problem

```
void solve(){
    f = 0;
    f_best = 1000000;
    for(int i = 1; i <= n; i++) appear[i] = 0;
    x[0] = 0;// starting point
    TRY(1);
    printf("%d",f_best);
}
int main(){
    input();
    solve();
}
```

Taxi Routing (TAXI)

- There are n passengers 1, 2, …, n. The passenger i want to travel from point i to point $i + n$ (i = 1,2,…,n). There is a taxi located at point 0 for transporting the passengers. You are given the distance matrix c in which $c(i,j)$ is the traveling distance from point i to point j (i, j = 0,1,…,2n). Compute the shortest route for the taxi, serving n passengers and coming back to point 0 such that at any moment, there are no more than one passenger in the taxi (the route visits each point 1, 2, …, 2n exactly once).

Taxi Routing

- Input:
  - Line 1 contains n ($1 \le n \le 11$)
  - Line i+1 (i = 1, 2,…, 2n+1) contains the i^{th} line of the matrix c
- Output:
  - Unique line contains the length of the shortest route

stdin	stdout
2 0 8 5 1 10 5 0 9 3 5 6 6 0 8 2 2 6 3 0 7 2 5 3 4 0	17

Taxi Routing

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

int n;// number of passengers (1,2,...,n). Passenger i has
        // pickup point i and drop-off point i + n
int c[2*MAX+1][2*MAX+1];// distance matrix
int cmin; // minimum distance between two distinct points
          // minimum element of the distance matrix (ignore
          // elements on the diagonal)

int x[MAX];
int appear[MAX];// appear[v] = 1 indicates that point i has
been visited
int load; // number of passenger in the taxi
int f;// accumulate distance of the route under construction
int f_best; // shortest distance found so far
```

Taxi Routing

```
void input(){
    scanf("%d",&n);
    cmin = 1000000;
    for(int i = 0; i <= 2*n;  i++){
        for(int j= 0; j <= 2*n; j++){
            scanf("%d",&c[i][j]);
            if(i != j && cmin > c[i][j])
             cmin = c[i][j];
        }
    }
}
```

Taxi Routing

```
int check(int v, int k){
    if(appear[v] == 1) return 0;
    if(v >n){
        if(!appear[v-n]) return 0;
    }else{
        if(load + 1 > 1) return 0;
    }
    return 1;
}
void solution(){
    if(f + c[x[2*n]][0] < f_best){
        f_best = f + c[x[2*n]][0];
    }
}
```

Taxi Routing

```
void TRY(int k){
    for(int v = 1; v <=2*n; v++){
        if(check(v,k)){
            x[k] = v;
            f += c[x[k-1]][x[k]];
            if(v <= n) load += 1; else load += -1;
            appear[v] = 1;
            if(k == 2*n) solution();
            else{
                if(f + (2*n+1-k)*cmin < f_best)
                    TRY(k+1);
            }
            if(v <= n) load -= 1; else load -= -1;
            appear[v] = 0;
            f -= c[x[k-1]][x[k]];
        }
    }
}
```

Taxi Routing

```
void solve(){
    load = 0;
    f = 0;
    f_best = 1000000;
    for(int i = 1; i <= 2*n; i++) appear[i] = 0;
    x[0] = 0;// starting point
    TRY(1);
    printf("%d",f_best);
}
int main(){
    input();
    solve();
}
```

Capacitated Bus Routing (CBUS)

- There are n passengers 1, 2, …, n. The passenger i want to travel from point i to point $i + n$ (i = 1,2,…,n). There is a bus located at point 0 and has k places for transporting the passengers (it means at any time, there are at most k passengers on the bus). You are given the distance matrix c in which $c(i,j)$ is the traveling distance from point i to point j (i, j = 0,1,…, 2n). Compute the shortest route for the bus, serving n passengers and coming back to point 0 (the route visits each point 1, 2, …, 2n exactly once).

Capacitated Bus Routing

- Input
  - Line 1 contains n and k ($1 \le n \le 11$, $1 \le k \le 10$)
  - Line i+1 (i = 1, 2,…, 2n+1) contains the (i-1)th line of the matrix c (rows and columns are indexed from 0,1,2,..,2n)
- Output:
  - Unique line contains the length of the shortest route

stdin	stdout
3 2 0 8 5 1 10 5 9 9 0 5 6 6 2 8 2 2 0 3 8 7 2 5 3 4 0 3 2 7 9 6 8 7 0 9 10 3 8 10 6 5 0 2 3 4 4 5 2 2 0	25

Capacitated Bus Routing

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

int n;// number of requests (1,2,...,n). Request i has
       // pickup point i and drop-off point i + n
int cap;// number of places of the bus
int c[2*MAX+1][2*MAX+1];// distance matrix
int cmin;

int x[MAX];
int appear[MAX];//appear[v] = 1 indicates v has appeared
int load;
int f;
int f_best;
```

Capacitated Bus Routing

```
void input(){
    scanf("%d%d",&n,&cap);
    cmin = 1000000;
    for(int i = 0; i <= 2*n;  i++){
        for(int j= 0; j <= 2*n; j++){
            scanf("%d",&c[i][j]);
            if(i != j && cmin > c[i][j])
             cmin = c[i][j];
        }
    }

}
```

Capacitated Bus Routing

```
int check(int v, int k){
    if(appear[v] == 1) return 0;
    if(v >n){
        if(!appear[v-n]) return 0;
    }else{
        if(load + 1 > cap) return 0;
    }
    return 1;
}
void solution(){
    if(f + c[x[2*n]][0] < f_best){
        f_best = f + c[x[2*n]][0];
    }
}
```

Capacitated Bus Routing

```
void TRY(int k){
    for(int v = 1; v <=2*n; v++){
        if(check(v,k)){
            x[k] = v;
            f += c[x[k-1]][x[k]];
            if(v <= n) load += 1; else load += -1;
            appear[v] = 1;
            if(k == 2*n) solution();
            else{
                if(f + (2*n+1-k)*cmin < f_best)
                    TRY(k+1);
            }
            if(v <= n) load -= 1; else load -= -1;
            appear[v] = 0;
            f -= c[x[k-1]][x[k]];
        }
    }
}
```

Capacitated Bus Routing

```
void solve(){
    load = 0;
    f = 0;
    f_best = 1000000;
    for(int i = 1; i <= 2*n; i++) appear[i] = 0;
    x[0] = 0;// starting point
    TRY(1);
    printf("%d",f_best);
}
int main(){
    input();
    solve();
}
```

Balanced Course Teacher Assignment (BCA)

- At the beginning of the semester, the head of a computer science department have to assign courses to teachers in a balanced way.
- The department has *m* teachers T={1, 2, ..., *m*} and *n* courses C={1, 2,..., *n*}.
- Each course c $\in$ C has a duration h_c.
- Each teacher t $\in$ *T* has a preference list which is a list of courses he/she can teach depending on his/her specialization.
- We know a list of pairs of conflicting two courses that cannot be assigned to the same teacher as these courses have been already scheduled in the same slot of the timetable. This conflict information is represented by a conflict matrix *A* in which A(*i*,*j*)=1 indicates that course *i* and *j* are conflict.
- The load of a teacher is the total duration of courses assigned to her/him.
- How to assign *n* courses to *m* teachers such that each course assigned to a teacher is in his/her preference list, no two conflicting courses are assigned to the same teacher, and the maximal load for all teachers is minimal.

Balanced Course Teacher Assignment

- Input
  - Line 1 contains n and m ($2 \le n \le 20$, $2 \le m \le 5$)
  - Line 2 contains $h_1,\ldots,h_n$
  - Line i+2 (i = 1,...,n) contains a positive integer k followed by k positive integers which are the teachers who can teach course i.
  - Line i+n+2 (i = 1,...,n) contains the i^{th} line of the conflict matrix A
- Output
  - The output contains a unique number which is the maximal load of the teachers in the solution found and the value -1 if not solution found.

stdin	stdout
4 2 3 7 2 1 2 1 2 2 1 2 2 1 2 2 1 2 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0	8

Balanced Course Teacher Assignment

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_N 50
#define MAX_M 10
// input data structures
int N;// number of classes
int M;// number of teachers
int sz[MAX_N];// sz[i] is the number of teachers for course i
int t[MAX_N][MAX_M];// t[c][i]: the ith teacher that can teach course c
int h[MAX_N];// h[c] is the number of hours of course c each week
int A[MAX_N][MAX_N];// A[i][j] = 1 indicates that course i and j are
                    // conflict
// variables
int X[MAX_N];// X[i] is the teacher assigned to course i
int f[MAX_M];// f[j] is the load of teacher j
int f_best;// best objective value
```

Balanced Course Teacher Assignment

```
void input(){
    scanf("%d%d",&N,&M);
    for(int i = 1; i <= N; i++)
        scanf("%d",&h[i]);
    for(int i = 1; i <= N; i++){
        scanf("%d",&sz[i]);
        for(int j = 0; j < sz[i]; j++)
            scanf("%d",&t[i][j]);
    }
    for(int i = 1; i <= N; i++){
        for(int j = 1; j <= N; j++)
            scanf("%d",&A[i][j]);
    }

}
```

Balanced Course Teacher Assignment

```
int check(int v, int k){
    for(int i = 1; i <= k-1; i++){
        if(A[i][k] && v == X[i]) return 0;
    }
    return 1;
}
void solution(){
    int max = 0;
    for(int i = 1;i <= M; i++){
        if(max < f[i]) max = f[i];
    }
    if(max < f_best){
        f_best = max;
    }
}
```

Balanced Course Teacher Assignment

```
void TRY(int k){// try to find a teacher (v) for course k
    for(int i = 0; i < sz[k]; i++){
        int v = t[k][i];
        if(check(v,k)){
                X[k] = v;
                f[v] += h[k];// accumulate load for teacher v
                if(k == N){
                    solution();
                }else{
                    TRY(k+1);
                }
                f[v] -= h[k];// recover load when backtracking
        }

    }
}
```

Balanced Course Teacher Assignment

```
void solve(){
    f_best = 1000000;
    for(int i = 1; i <= M; i++) f[i] = 0;
    TRY(1);
    if(f_best == 1000000)// no solution found
       printf(“-1”);
    else
       printf("%d",f_best);
}
int main(){
    input();
    solve();
}
```

Balanced Academic Curriculum Problem (BACP)

- The BACP is to design a balanced academic curriculum by assigning periods to courses in a way that the academic load of each period is balanced .
- There are N courses 1, 2, …, N that must be assigned to M periods 1, 2, …, M. Each course *i* has credit c_i and has some courses as prerequisites. The load of a period is defined to be the sum of credits of courses assigned to that period.
- The prerequisites information is represented by a matrix A_{NxN} in which $A_{i,j} = 1$ indicates that course *i* must be assigned to a period before the period to which the course *j* is assigned. Given constants a, b. Compute the solution satisfying constraints
  - Total number of courses assigned to each period is greater or equal to a and smaller or equal to b
  - The maximum load for all periods is minimal

Balanced Academic Curriculum Problem (BACP)

- Input
  - Line 1 contains N and M ($2 \leq N \leq 16$, $2 \leq M \leq 5$)
  - Line 2 contains $c_1, c_2, \ldots, c_N$
  - Line i+2 (i = 1,…,N) contains the i^{th} line of the matrix A
- Output
  - Unique line contains that maximum load for all periods of the solution found

Balanced Academic Curriculum Problem (BACP)

```
#include <bits/stdc++.h>
#define MAX_N 30
#define MAX_M 10

// input
int N, M;
int c[MAX_N];// c[i] is the credits of course i
int A[MAX_N][MAX_N];
// solution representation
int x[MAX_N];// x[c] is the period to which the course c is
             // assigned
int f_best;
int load[MAX_M];// load [p] is the load of period p
```

Balanced Academic Curriculum Problem (BACP)

```
int check(int v, int k){
    for(int i = 1; i <= k-1; i++){
        if(A[i][k] == 1){
                if(x[i] >= v) return 0;
        } else if(A[k][i] == 1){
            if(v >= x[i]) return 0;
        }
    }
    return 1;
}
void solution(){
    int max = load[1];
    for(int i = 2; i <= M; i++) max = max < load[i] ? load[i] : max;
    if(max < f_best){
        f_best = max;
    }
}
```

Balanced Academic Curriculum Problem (BACP)

```
void TRY(int k){
    for(int v = 1; v <= M; v++){
        if(check(v,k)){
                x[k] = v;
                load[v] += c[k];
                if(k == N) solution();
                else TRY(k+1);
                load[v] -= c[k];
        }
    }
}
void input(){
    scanf("%d%d",&N,&M);
    for(int i = 1; i <= N; i++)  scanf("%d",&c[i]);
    for(int i = 1; i <= N; i++)
        for(int j = 1; j <= N; j++)  scanf("%d",&A[i][j]);
}
```

Balanced Academic Curriculum Problem (BACP)

```
void solve(){
    for(int i = 1; i <= M; i++) load[i] = 0;
    f_best = 1000000;
    TRY(1);
    printf("%d\n",f_best);
}
int main(){
    input();
    solve();
}
```

Paper Reviewers Assignment Problem (PRAP)

- The chair of a conference must assign scientific papers to reviewers in a balance way. There are *N* papers 1, 2, …, *N* and *M* reviewers 1, 2, …, *M*.
- Each paper *i* has a list *L*(*i*) of reviewers who are willing to review that paper.
- A review plan is an assignment reviewers to papers. The load of a reviewer is the number of papers he/she have to review.
- Given a constant b, compute the assignment such that
  - Each paper is reviewed by exactly b reviewers
  - The maximum load of all reviewers is minimal

Paper Reviewers Assignment Problem (PRAP)

- Input
  - Line 1 contains N, M and b
  - Line i+1 (i = 1,…,N) contains a positive integer k followed by k positive integers representing the list L(i)
- Output
  - Unique line contains the maximum load for all reviewers of the solution found or contains -1 if no solution found.

Paper Reviewers Assignment Problem (PRAP)

```
#include <stdio.h>
#define MAX_N 20
#define MAX_M 10
#define MAX_B 10
int N,M,b;
int sz[MAX_N+1];
int A[MAX_N+1][MAX_M+1];
int X[MAX_N+1][MAX_B+1];// X[k][i] is the ith reviewers of paper k
int load[MAX_M+1];// load[i] is the number of papers assigned to reviewer i
int f_best;

void input(){
    scanf("%d%d%d",&N,&M,&b);
    for(int i = 1; i <= N; i++){
        scanf("%d",&sz[i]);
        for(int j = 0; j < sz[i]; j++)
            scanf("%d",&A[i][j]);
    }
}
```

Paper Reviewers Assignment Problem (PRAP)

```
void solution(){
    int max_load = load[1];
    for(int i = 2; i <= M; i++){
        if(max_load < load[i]) max_load = load[i];
    }
    if(max_load < f_best){
        f_best = max_load;
    }
}
int check(int v,int k, int i){
    if(load[k] >= f_best-1) return 0;
    return v > X[k][i-1];
}
```

Paper Reviewers Assignment Problem (PRAP)

```
void TRY(int k, int i){
    for(int j = 0; j < sz[k]; j++){
        int v = A[k][j];
        if(check(v,k,i)){
            X[k][i] = v;
            load[v]++;
            if(k == N){
                if(i == b) solution();
                else TRY(k,i+1);
            }else{
                if(i == b) TRY(k+1,1);
                else TRY(k,i+1);
            }
            load[v]--;
        }
    }
}
```

Paper Reviewers Assignment Problem (PRAP)

```
void solve(){
    f_best = 1000000;
    for(int i = 1; i <= M; i++){ load[i] = 0; X[i][0] = 0;}
    TRY(1,1);
    printf("%d",f_best);
}

int main(){
    input();
    solve();
}
```

Vehicle Routing Problem (VRP)

- A fleet of K identical trucks having capacity Q need to be scheduled to delivery pepsi packages from a central depot 0 to clients 1, 2, …, n. Each client i requests $d[i]$ packages. The distance from location i to location j is $c[i, j]$, $\forall\ 0 \leq i, j \leq n$.
- Solution: For each truck, a route from depot, visiting clients and returning to the depot for deliverying requested pepsi packages such that:
  - Each client is visited exactly by one route
  - Total number of packages requested by clients of each truck cannot exceed its capacity
- Goal: List all solutions

Vehicle Routing Problem (VRP)

- Input
  - Line 1: n, K, Q
  - Line 2: d[1], . . ., d[n]
  - Line *i* + 3: the *i*th row of the distance matrix *c* (i = 0, …, n)
- Output
  - All solutions

Vehicle Routing Problem (VRP)

```
#include <stdio.h>
#define MAX 50
int n,K,Q;
int d[MAX];
int c[MAX][MAX];

int x[MAX];// x[i] is the next point of i (i = 1,...,n), x[i]
           // in {0,1,...,n}
int y[MAX];// y[k] is the start point of route k
int load[MAX];
int visited[MAX];// visited[i] = 1 means that client point i
has been visited
int segments;// number of segments accumulated
int nbRoutes;
```

Vehicle Routing Problem (VRP)

```
void input(){
    scanf("%d%d%d",&n,&K,&Q);
    for(int i = 1; i <= n; i++){
        scanf("%d",&d[i]);
    }
    d[0] = 0;
    for(int i = 0; i <= n; i++){
        for(int j = 0; j <= n; j++){
            scanf("%d",&c[i][j]);
        }
    }
}
```

Vehicle Routing Problem (VRP)

```
void solution(){
   for(int k = 1; k <= K; k++){
        int s = y[k];
        printf("route[%d]:  0 ",k);
        for(int v = s; v != 0; v = x[v]){
            printf("%d ",v);
        }
        printf("0\n");
    }
    printf("--------------------------\n");
}
```

Vehicle Routing Problem (VRP)

```
int checkX(int v,int k){
    if(v > 0 && visited[v]) return 0;
    if(load[k] + d[v] > Q) return 0;
    return 1;
}
int checkY(int v, int k){
    if(v == 0) return 1;
    if(load[k] + d[v] > Q) return 0;
    return !visited[v];
}
```

Vehicle Routing Problem (VRP)

```
void TRY_X(int s, int k){
    if(s == 0){
        if(k < K) TRY_X(y[k+1],k+1);
        return;
    }
    for(int v = 0; v <= n; v++){
        if(checkX(v,k)){
            x[s] = v;  visited[v] = 1;  load[k] += d[v]; segments++;
            if(v > 0) TRY_X(v,k);
            else{
                if(k == K){
                        if(segments == n+nbRoutes) solution();
                }else  TRY_X(y[k+1],k+1);
            }
            segments--;  load[k] -= d[v];  visited[v] = 0;
        }
    }
}
```

Vehicle Routing Problem (VRP)

```
void TRY_Y(int k){
    for(int v = (y[k-1]==0 ? 0 : y[k-1] + 1); v <= n; v++){
        if(checkY(v,k)){
            y[k] = v; visited[v] = 1; load[k] += d[v];
            if(v > 0) segments += 1;
            if(k < K) TRY_Y(k+1);
            else{
                nbRoutes = segments;
                TRY_X(y[1],1);
            }
            load[k] -= d[v]; visited[v] = 0;
            if(v > 0) segments -= 1;
        }
    }
}
```

Vehicle Routing Problem (VRP)

```
void solve(){
    for(int v = 1; v <= n; v++) visited[v] = 0;
    y[0] = 0;
    TRY_Y(1);
}
int main(){
    input();
    solve();
}
```