Korešpondenčný seminár z programovania
XXIII. ročník, 2005/06
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporujú: aSc – Applied Software Consultants spol. s r.o.
MICROSTEP-MIS spol. s r.o.

Vzorové riešenia 1. kola zimnej časti

Zima sa nám pomaličky blíži, aj prvý sneh sme tu už mali... Nás konkrétne stretol v Budapešti, kde sme si boli aj my zaprogramovať (na regionálnom kole veľkej svetovej súťaže ACM ICPC) – a hanbu sme si teda nespravili. A hanbu ste si nespravili ani vy, väčšina riešení prvého kola nám dávala zmysel `:-)` a bodov ste si nahonobili dosť a dosť. A ak sa vám máli, šup šup čítať tieto riešenia, keď ich prečítate, nabudúce budete múdrejší. Alebo aspoň sčítanejší `;-)`

Povedz dvakrát „povedz dvakrát povedz dvakrát“

KSPáci

1. O Kingu

opravoval Stano
(max. 15 bodov)

Nuž, zamyslime sa, ako by sme mohli riešiť tento príklad. Začnime najľahším riešením, ktoré potrebuje čas $O(NK)$ – v pomocnom poli si budeme udržovať K doteraz najväčších čísel. (Je jedno, či ich mám utriedené, alebo nie.) Nuž, veľmi pomalé.

Dosť veľa ľudí prišlo na to, že to isté vieme spraviť s prioritnou frontou (haldou) v čase $O(N \log K)$. Rovnakú možnosť nám ponúkajú aj napríklad vyvážené binárne stromy. Väčšinu už takéto riešenie uspokojilo. V zadaní ale nebola len tak pre nič-za nič veta, že keď je vaše riešenie príliš jednoduché, asi nie je optimálne.

Ako na lepšie riešenie? Naše pole si „nasekáme“ na úseky dĺžky K (posledný bude možno kratší) a postupne po jednom budeme tieto úseky spracúvať. Predstavme si teraz, že sme už spracovali niekoľko úsekov, a teda máme K doteraz najväčších čísel. Prečítame ďalší úsek čísel zo vstupu. Tým sme dostali $2K$ čísel, spomedzi ktorých chceme nájsť K najväčších. Na to nám stačí nájsť K-te najväčšie, potom už bude všetko jasné.

Ako na to? Riešenie prichádza vo forme algoritmu známeho pod menom quickmedian. Tento algoritmus je podobný algoritmu quicksort, ktorý určite všetci dobre poznáte. Na začiatku si zvolíme pivota (k jeho výberu neskôr), podľa ktorého preusporiadame všetky prvky. Na jednu stranu menšie od neho, na druhú stranu väčšie.

Takže vieme že na jednej strane je x prvkov, na druhej $N-1-x$. Rozdiel oproti quicksortu je ten, že nechceme celé pole utriediť, zaujíma nás len to, ktorý prvok bude na K-tom mieste. V tomto okamihu však vieme povedať, na ktorej strane pivota K-ty prvok leží, no a rekurzívne sa zavoláme len na tú stranu. Takýmto spôsobom vyčleníme vždy nejaké prvky, ktoré nás už odvtedy nezaujímajú.

V priemernom prípade je quickmedian lineárny od veľkosti poľa – každým výberom pivota sa v priemere zbavíme jednej polovice prvkov. Podobne ako quicksort, aj quickmedian je v najhoršom prípade až kvadratický. Pomoc je rovnaká, ako u quicksortu, možností je viac:

1. Vyberieme pivota náhodne.
2. Vyberieme náhodne troch pivotov, použijeme prostredného z nich.
3. V lineárnom čase nájdeme vhodného kandidáta na pivota. Jednou možnosťou je známy median-of-five partitioning: Pole rozdelíme na pätice, každej pätici nájdeme v konštant-

nom čase medián, rekurzívnym volaním nájdeme medián týchto mediánov a ten použijeme ako pivota na rozdelenie pôvodného poľa.

Po každom načítaní bloku K čísel zo vstupu teda spustíme na aktuálnych $2K$ kandidátov quickmedian, K najväčších si zapamätáme a ideme ďalej. Každé volanie quickmedianu je v čase $O(K)$, dokopy sa nám to nasčíta na $O(N)$. Našli sme teda lineárny algoritmus a lepšie to už zjavne nepôjde.

Listing programu:

```
#include <cstdio>
#include <iostream>
using namespace std;

#define MAX 4747
int n,k,data[MAX];

void vyber(int left,int right){
  if (left>k || right<=k) return;  // ak K lezi mimo intervalu, skoncime
  if (left+1==right) return;   // podobne, interval dlzky 1 je uz utriedeny

  int middle=data[(left+right)/2];  // zvolime pivota
  int l=left,r=right-1;
  while(l<=r) {
    while (data[l]>middle) l++;
    while (data[r]<middle) r--;
    if (l<=r){int x=data[l]; data[l]=data[r];data[r]=x;l++;r--;}
  }
  // rekurzivne sa zavolame na obe casti, ta, v ktorej nie je K, rovno skonci
  vyber(left,r+1);
  vyber(l   ,right);
}

int main(){
  int q=0;
  scanf("%d %d ",&n,&k);
  while(q<n){
    int i=0;
    while (q<n && i<k) {
      scanf("%d ",&data[k+i++]);
      q++;
    }
    vyber(0,k+i);
  }
  for(int i=0;i<k;i++) printf("%d ",data[i]);
}
```

2. O šifrochtivých KSPákoch

opravoval Mic
(max. 15 bodov)

Tento príklad bol veľmi jednoduchý a preto ste všetci zistili (česť výnimkám) správnu stratégiu. Akurát s implementáciou to bolo horšie a navyše neviem prečo drvivá väčšina z Vás predpokladala že vstup je utriedený, aj keď to tam nikde nebolo napísané. Tak poďme najskôr k bodovaniu:

- $O(N \log N)$ – za vzorové riešenie sa dalo získať 15 bodov,
- $O(N^2)$ – 12 bodov,

- *pomalšie* – 10 bodov,
- za predpoklad že vstup je utriedený som to rátal ako o rád horšie riešenie,
- za zlý odhad zložitosti som strhával 1 bod,
- za chyby v riešeniach som dával bonus −1 bod,
- za nedostatočný popis extra bonus −1 bod,
- za pekné obrázky v riešení som udeľoval ⋆.

Idea riešenia bola veľmi jednoduchá. Pozriem sa na všetky benzínky v mojom dosahu a natankujem po najbližšiu lacnejšiu. Ak taká neexistuje (teda všetky benzínky v mojom dosahu majú drahší benzín ako na benzínke na ktorej som), tak natankujem plnú nádrž a posuniem sa na najlacnejšiu benzínku v mojom dosahu a tam sa ďalej rozhodujem ako budem tankovať. Ale je táto stratégia naozaj najlepšia? Skúsme si to ukázať.

V prvom prípade je rozhodnutie zjavne optimálne. Nech som na benzínke A, najbližšia lacnejšia je B a je v mojom dosahu. Tankovať viac ako po B sa mi neoplatí – to, čo teraz natankujem navyše, by som mohol rovnako dobre natankovať až na B, a to lacnejšie. Podobne, ak by som natankoval menej ako po B, musel by som cestou k B ešte tankovať, a to aspoň tak drahý benzín.

V druhom prípade je zjavné, že sa mi oplatí nakúpiť plnú nádrž. Nech teraz je C najlacnejšia benzínka v mojom dojazde. Čokoľvek, čo kúpim cestou na C, si môžem kúpiť až tam, a opäť raz platí, že na tom neprerobím. Preto sa stačí ďalej rozhodovať až na C.

Táto stratégia nám dáva už jedno celkom dobré riešenie. Utriedime si benzínky. Začneme na matfyze, a nájdeme najbližšiu lacnejšiu a tam sa posunieme, alebo sa posunieme na najlacnejšiu v dosahu. Toto nám dáva riešenie kvadratické, čiže $O(N^2)$ riešenie.

Ale čo tam vlastne robíme? Neustále hľadáme minimum z nejakej fronty benzíniek (vždy, keď sa nová benzínka dostane do dosahu, zaradím ju do fronty, keď okolo nejakej prejdem, z fronty ju vyhodím). Poďme na to hľadanie minima šikovnejšie.

Naprogramujeme si takú špeciálnu dátovu štruktúru, do ktorej budeme hádzať benzínky, vyberať ich (nie nutne v tom poradí, v ktorom sme ich vkladali!) a budeme vedieť zistiť, ktorá benzínka v nej je najlacnejšia. Táto štruktúra bude mat nasledovne vlastnosti:

1. Benzínky, ktoré sú v nej, sú v poradí, v akom sme ich vkladali.
2. Benzínky, ktoré sú v nej, sú usporiadané od najlacnejšej po najdrahšiu.

Ako to implementovať? Použijeme deque (čítaj „dek“, je to double-ended queue, teda fronta s dvoma koncami). Budeme potrebovať nasledujúce operácie: „odober benzínku zo začiatku“, „odober benzínku z konca“ a „pridaj benzínku na koniec“. Jediný rozdiel oproti normálnej fronte bude taký, že vždy, keď ideme pridať novú benzínku, najskôr vyhodíme z konca deque všetky, ktoré sú od nej drahšie.

No ale ako nám toto pomôže k riešeniu? Predstavme si, že sme na nejakej benzínke. Najskôr povyhadzujeme z deque tie benzínky, ktoré sme už prešli (aj tú, na ktorej stojíme). Teraz by sme sa chceli posunúť na najbližšiu lacnejšiu benzínku v dosahu. Ak je prvá benzínka v deque lacnejšia ako tá, kde práve sme, ideme na ňu. Ak nie, pridávame do štruktúry nasledujúce benzínky, ktoré sú v dosahu (a iné pritom občas vyhodíme). Toto robíme, kým sme nepridali všetky v dosahu alebo kým nenájdeme nejakú lacnejšiu od tej, kde sme. Ak sme lacnejšiu benzínku v dosahu nenašli, posunieme sa na najlacnejšiu v dosahu, čo je opäť prvá v našej deque. Takže na tú sa môžem posunúť a tam rovnako pokračovať.

Formálny dôkaz, že to funguje, by vyzeral asi nasledovne: Dokážeme (matematickou indukciou od čísla benzínky, pri ktorej sme), že všetky ďalšie benzínky, pri ktorých sa nám ešte oplatí zastať, sú buď v deque, alebo ešte neboli spracované.

Jediný zaujímavý krok je, keď práve vyhadzujeme nejakú benzínku z konca deque. Potrebujeme dokázať, že ju v optimálnom riešení nepoužijeme. Určite už nikdy nebude „najlacnejšia v dosahu“, lebo dôvodom, že ju vyhadzujeme, je, že práve máme v dosahu inú, lacnejšiu.

Všetky benzínky, ktoré ležia medzi nami a ňou, sme už spracovali, takže vieme, že na tých z nich, ktoré nie sú v deque, zastávať nebudeme. Ale tie, ktoré v deque sú, sú pred ňou, a teda je na nich lacnejší benzín. Preto ani pravidlom o „najbližšej lacnejšej“ sa na ňu nikdy nedostaneme. Howgh.

A ako je to s časovou zložitosťou? Tvrdím že je $O(N)$. A prečo? Máme N benzínok a každú do deque práve raz vložím, a raz vyberiem. A ešte maximálne N krát budem zisťovať najmenší prvok. A keďže všetky tieto operácie bežia v konštantnom čase, tak aj celková zložitosť algoritmu bude lineárna. Akurát je tu ešte jeden zádrhel. Kedže vstup nie je utriedený, tak sa celková zložitosť algoritmu zhorší na $O(N \log N)$.

Iné riešenie, by MišoF. Predstavte si, že by sme dostali možnosť benzín hocikedy predať za cenu, za ktorú sme ho kúpili. Hodnotu optimálneho riešenia to nezmení. V takomto prípade je už riešenie ľahké. Vždy, keď sa hýbeme, používame najlacnejší benzín, čo máme práve k dispozícii. Zastavíme na každej benzínke, vždy predáme všetok benzín, čo ešte máme v nádrži a je drahší ako tu predávajú[1] *a následne natankujeme doplna. Aj toto riešenie sa dá ľahko naprogramovať použitím deque a je podľa mňa o dosť očividnejšie, že naozaj funguje, nie?*

Listing programu:

```
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;

#define MAXN 100000

typedef pair<int,int> PII;
// to iste ako struct PII { int first,second; };
// a navyse to ma operator <, ktory porovnava najskor podla first, potom podla second

vector<PII> benzinky;

PII buf[MAXN];
int beg,end;

void push(PII p)
  { while(beg!=end && buf[end-1].second>p.second) end--;  buf[end]=p;  end++; }
void pop(){ beg++; }
PII minimum(){ return buf[beg]; }
bool empty(){ return beg==end; }

int main(){
    int N,D,C,K;
    scanf("%d %d %d %d",&N,&D,&C,&K);
    for(int i=0;i<N;i++){
        int a,b;
        scanf("%d %d",&a,&b);
        benzinky.push_back(PII(a,b));
    }
    //pridame si este cielovu benzinku, a na nej tankujeme zadarmo
    benzinky.push_back(PII(D,0));
    //Vec na ktoru ste takmer vsetci zabudli
    sort(benzinky.begin(),benzinky.end());
```

[1] *Riešenie pôvodnej úlohy dostaneme tak, že by sme si tento nepoužitý benzín ani nekúpili.*

```
    int v=0,s=0,k=0;
    long long sum=0;
    beg=end=0;
    while(v!=D){
        while(!empty() && minimum().first<=v) pop();
        while(s<=N && (minimum().second>C || empty()) && benzinky[s].first<=v+K)
            push(benzinky[s++]);
        if(minimum().second<C){
            int vzd=minimum().first-v;
            sum+=C*(vzd-k);
            C=minimum().second;
            v+=vzd;
            k=0;
        }else{    //tankujem plnu nadrz
            if(minimum().first==v) pop();
            if(empty()) { printf("Neda sa dostat na hru.\n"); return 0; }
            sum+=(K-k)*C;
            C=minimum().second;
            k=K+v-minimum().first;
            v=minimum().first;
        }
    }
    printf("%lld\n",sum);
    return 0;
}
```

opravoval Lukáš
(max. 15 bodov)

3. Obrovská šachovnica

Tento príklad nepatril medzi ťažké. Najprv si ukážeme riešenie v čase $O((VS)^2)$. Pre všetky príšery zistíme miesto, kde sa majú jazdi stretnúť tak, aby najviac jeden z nich stúpil na danú príšeru. Konkrétne pre každého jazdca spravíme dve prehľadávania do šírky. Prvé bude také, že nemôže stúpiť na žiadnu príšeru, v druhom už môže stúpiť na danú príšeru. Potom na každom políčku vyberieme jazdca, ktorý si stúpením na príšeru skrátil cestu najviac a spočítame vzdialenosť, ktorú musia dokopy prejsť. Vyberieme políčko, ktoré má minimálny súčet vzdialeností. Keďže príšer môže byť veľmi veľa, toto riešenie nie je optimálne.

Teraz si ukážeme optimálne riešenie v čase $O(VS)$. Pre každého jazdca prehľadáme šachovnicu raz, pričom pre každé políčko si pamätáme dva stavy. Nultý stav hovorí, že sa na políčko vieme dostať bez stúpenia na príšeru, prvý hovorí, že sa doňho vieme dostať s jedným stúpením na príšeru. Na šachovnici si vytvoríme graf, pričom hrana medzi dvoma políčkami pôjde práve vtedy, ak sa z jedného políčka na druhé vie dostať jazdec. Hrana z políčka so stavom 0 ide na iné políčko so stavom 1 práve vtedy, keď na druhom políčku je príšera. Inak ide hrana medzi týmito políčkami zo stavu s do stavu s ($s \in \{0, 1\}$).

Pre nejakého jazdca začneme graf prehľadávať do šírky z políčka, kde stojí na začiatku a so stavom 0, lebo ešte neskočil na žiadnu príšeru. Prehľadaním grafu nájdeme najkratšie vzdialenosti do všetkých vrcholov, teda pre každé políčko vieme povedať, akú vzdialenosť musí prejsť jazdec bez jediného stúpenia na príšeru a s jedným stúpením na príšeru.

Po vykonaní štyroch prehľadávaní stačí len prejsť celú šachovnicu a pre každé políčko zrátať, koľko musia jazdci prejsť, aby sa stretli. Máme dve možnosti – nikto nestúpil na príšeru alebo nejaký jazdec stúpil na príšeru. V druhom prípade vyberieme toho jazdca, ktorému stúpenie na príšeru skráti cestu najviac.

Vykonáme 4 prehľadávania do šírky a nakoniec prejdeme celú šachovnicu, preto časová zložitosť je $O(VS)$. Pamätáme si celú šachovnicu, takže pamäťová zložitosť je tiež $O(VS)$.

Ešte pár slov k bodovaniu. Za vzorové riešenie bolo ako obvykle 15 bodov, za riešenie v čase $O((VS)^2)$ 11 bodov a za pomalšie ešte menej. Za chýbajúci program som strhal polovicu bodov a nejaké body ste mohli stratiť aj za malé chyby v programe.

Listing programu:

```
#include <iostream>
#include <queue>
using namespace std;

int vz[100][100][4][2], v, s, n;
int prisery[100][100];
typedef pair<int, int> DVA;
typedef pair<DVA, int> TRI;
int h[]={1, -1, -2, -2, -1, 1, 2, 2};
int g[]={2, 2, 1, -1, -2, -2, -1, 1};
//všetkých osem smerov pohybu

void porataj(int x, int y, int t) {
   vz[x][y][t][0]=0;
   queue<TRI> f; f.push(TRI(DVA(x, y), 0));
   //vrchol vo fronte určujú tri čísla, prvé dve sú súradnice, tretie je stav

   for (; !f.empty(); f.pop()) {
      DVA u=f.front().first, w;
      int stav=f.front().second, novy_stav;

      for (int i=0; i<8; i++) {
         w=u; w.first+=h[i]; w.second+=g[i];
         if (w.first<0 || w.first>=v || w.second<0 || w.second>=s) continue;
         //skáčeme mimo šachovnice

         novy_stav=stav+prisery[w.first][w.second];
         if (novy_stav>1) continue;//dvakrát by sme stúpili na príšeru

         if (vz[w.first][w.second][t][novy_stav]==100000000) {
            vz[w.first][w.second][t][novy_stav]=
               vz[u.first][u.second][t][stav]+1;

            f.push(TRI(w, novy_stav));
         }
      }
   }
}

int main() {
   cin>>v>>s;
   int x[4], y[4];
   for (int i=0; i<4; i++) cin >> x[i] >> y[i];
   for (int i=0; i<v; i++) for (int j=0; j<s; j++) prisery[i][j]=0;

   cin >> n;
   for (int i=0; i<n; i++) {
      int c, d; cin >> c >> d;
      c--; d--;
      prisery[c][d]=1;
   }

   for (int i=0; i<4; i++) {
      x[i]--; y[i]--;
```

```
        for (int k=0; k<v; k++) for (int l=0; l<s; l++)
            for (int o=0; o<2; o++) vz[k][l][i][o]=100000000;
        porataj(x[i], y[i], i);
    }

    int min=100000000;
    for (int i=0; i<v; i++) for (int j=0; j<s; j++) {
        int suma=0, t;
        for (int k=0; k<4; k++) suma+=vz[i][j][k][0];

        t=s;
        for (int k=0; k<4; k++)
            if (suma+vz[i][j][k][1]-vz[i][j][k][0]<t)
                t=suma+vz[i][j][k][1]-vz[i][j][k][0];

        if (t<min) min=t;
    }

    if (min>=100000000) cout<<"Nemozu sa stretnut."<<endl;
    else cout<<min<<endl;
}
```

4. O poriadnych a výnimočných kockách

opravoval Bee
(max. 15 bodov)

Riešenia tohto príkladu sa dajú rozdeliť na dve skupiny a síce také, ktoré využívali grafy a také, ktoré nie. Tie, ktoré ich nevyužívali, boli buď zlé (maximálne 4 body) alebo backtracky (maximálne 8 bodov). Všetky tie, ktoré grafy využívali, boli v čase $O(N(N+M))$ (N je počet vrcholov a M je počet hrán), a teda dostali 15 bodov.

Takže si poďme povedať, odkiaľ sa nám v tomto príklade zobral graf. A nie len tak hocijaký, ale bipartitný. To je taký graf, ktorý ma dve skupiny vrcholov A a B, pričom všetky hrany, ktoré v tomto grafe sú, musia spájať jeden vrchol z A s jedným vrcholom z B. Inak povedané, medzi žiadnymi dvoma vrcholmi z A nie je hrana, to isté platí aj pre B. V našom príklade vrcholy v skupine A budú zodpovedať písmenkám slova, ktoré skladáme, vrcholy skupiny B tvoria kocky, ktoré máme k dispozícii. Hrana medzi kockou a písmenkom existuje pravé vtedy, ak je na kocke to písmenko.

No dobre, ale čo teraz s takýmto grafom? Odpoveď je ľahká: chceme nájsť také hrany, aby každé písmenko bolo spojene práve s jednou kockou a každá kocka s najviac jedným písmenkom. Inými slovami, chceme nájsť čo najväčšiu množinu hrán, v ktorej žiadne dve hrany nemajú spoločný vrchol.[2] Táto úloha je v teórii grafov známa ako hľadanie maximálneho (perfektného) párovania. Existuje množstvo algoritmov, ktorými sa dá riešiť, a niektoré majú časovú zložitosť lepšiu ako naše vzorové riešenie, ale my sme sa radšej rozhodli uviesť taký algoritmus, ktorý máte šancu pochopiť `:-)`.

Takže ako na to? Napodiv to nebude nič zložité. Predstavme si, že sme už našli nejakých k hrán, tieto budeme volať párovacie. Radi by sme buď našli ďalšiu, alebo vyhlásili, že sa to nedá. Na to si ale musíme zaviesť nový pojem – zlepšujúca cesta. Je to cesta, ktorá začína v písmenku, skladá sa striedavo z nepárovacích a párovacích hrán, pričom nepárovacou hranou začína aj končí. Ľahko vidieť, že ak takúto cestu v grafe nájdeme, tak nám stačí vymeniť jej párovacie hrany za nepárovacie a naopak – a keďže nepárovacích bolo o jednu viac, tak sme práve našli párovanie veľkosti $k+1$.

[2] A keď ju nájdeme, pozrieme sa, či z každého písmenka ide nejaká z vybratých hrán.

Zostáva ukázať, že ak už zlepšujúcu cestu nenájdeme, je aktuálne párovanie najväčšie možné. V prvom rade si rozmyslite, že toto nie je zjavné! Vieme totiž zatiaľ len to, že našim postupom toto naše párovanie nevieme zlepšiť. Nevieme nič o tom, či sa iným, lepším spôsobom nedá vyrobiť lepšie párovanie.

Nuž, nedá. Zoberme si dve párovania, z ktorých jedno je maximálne a druhé je od neho menšie. Pozrime sa na graf, ktorý tvoria tie hrany, ktoré sú v práve jednom z týchto párovaní. Za domácu úlohu si rozmyslite, že aspoň jeden z komponentov tohto grafu zodpovedá zlepšujúcej ceste pre to menšie párovanie. Z toho vyplýva, že z **ľubovoľného** párovania viem pomocou zlepšujúcich ciest dostať maximálne – a teda ak už zlepšujúca cesta neexistuje, párovanie je nutne najväčšie možné.

Poďme si teraz presnejšie popísať náš algoritmus. Použijeme upravené prehľadávanie do hĺbky, pretože pri návrate z rekurzie môžeme pohodlne prehodiť párovacie hrany za nepárovacie a naopak. Finta navyše: k-tu zlepšujúcu cestu začneme hľadať len z k-teho písmenka. Po k-tej fáze teda buď máme nejako spárovaných prvých k písmen, alebo sme už zistili, že sa to nedá – a teda nejde poskladať ani celé slovo.

Keď sme v nejakom písmenku, pozrieme sa na všetky kocky s ktorými susedí. Ak je niektorá z nich nespárovaná, spárujeme ju s naším písmenkom a vrátime nejakú hodnotu, ktorá bude signalizovať, že sa našla zlepšujúca cesta a máme poprehadzovať hrany. Ak narazíme na spárovanú kocku, tak sa zavoláme na písmenko, s ktorou je spárované. Ak nám to písmenko vráti, že sa našla zlepšujúca cesta, tak tú kocku, s ktorou bolo spárované, spárujeme s naším písmenkom (všimnite si, že týmto ju odpárujeme od pôvodného písmenka, čím efektívne prehadzujeme hrany) a tiež vrátime signál o nájdení zlepšujúcej cesty. Ak nám ale všetky susedné (v zmysle párovania s kockami) písmenka vrátia, že sa cesta nenašla, tak vrátime aj my, že sa nenašla. Takto teda v čase $O(N + M)$ nájdeme zlepšujúcu cestu pre jedno písmenko. Ak toto prehľadávanie zavoláme pre každé písmenko, tak nám stačí skontrolovať, že či sa pre každé vrátil signál nájdenia zlepšujúcej cesty. Ak nie, môžeme skončiť. Celkovo ho teda voláme $O(N)$-krát a dokopy nám to dáva sľúbených $O(N(N + M))$.

Listing programu:

```
#include<cstdio>
#include<cstring>

const int MAX = 100;
int spar[MAX]; // sparovane kocky
int vis[MAX]; // navstivene pismenka
int graf[MAX][MAX]; // graf[pismenko][kocka]
int n, l; // pocet kociek, pocet pismeniek

void init() {
  char text[MAX];
  scanf("%s ", text);
  l = strlen(text);
  scanf("%d ", &n);
  memset(graf, 0, n*l);
  for(int i=0; i<n; i++)
  {
    spar[i] = -1;
    static char s[6];
    scanf("%s ", s);
    for(int j=0; j<l; j++)
      for(int k=0; k<6; k++)
        graf[j][i] = (text[j] == s[k] || graf[j][i]);
  }
}
```

```
int zlepsi(int tu) {
  vis[tu] = 1;
  for(int i=0; i<n; i++) // prezerame vsetky kocky...
    if(graf[tu][i]) // ...s ktorymi je spojeny prehladavany vrchol 'tu'
    {
      if(spar[i] == -1) // nasli sme nesparenu kocku:
      {
        spar[i] = tu; // sparime ju s 'tu'
        return 1; // vratime signal
      }
      else
        if(!vis[spar[i]]) // ak sme doteraz este i-tu kocku nenavstivili...
          if(zlepsi(spar[i])) // ...a nasli sme zlepsujucu cestu cez kocku i:
          {
            spar[i] = tu; // tak odparime kocku i od pismenka spar[i] a sparime ju s 'tu'
            return 1; // vratime signal
          }
    }
  return 0; //nenasla sa zlepsujuca cesta
}

int main() {
  init();
  bool match = 1;
  for(int i=0; i<l; i++)
  {
    for(int j=0; j<l; j++) vis[j] = 0;
    if(!zlepsi(i)){ match=0; break; } // pismenko sa neda sparovat, mozeme skoncit
  }
  if(match) printf("Da sa\n"); else printf("Neda sa\n");
}
```

opravoval MišoF.
(max. 15 bodov bodov)

5. Optimálna sada mincí

Mno, dosť toho dorazilo, dosť, viac, než som čakal. Kvalita už miestami očakávaniu zodpovedala (5 riešení bolo písaných rukou na papier, výpočtovú silu počítača teda nijak nevyužili... a na ich výsledku sa to aj odrazilo).

No ale žiadne kecy, pome k veci[3]!

V prvom rade sa zamyslime, ako čo najefektívnejšie určiť pre každú sumu od 1 do N, na koľko najmenej mincí ju vieme zaplatiť. V prvom rade si uvedomme, že pri platení ľubovoľnej z týchto súm si s predavačom vieme odovzdávať jednu mincu po druhej tak, aby aktuálne zaplatená suma nikdy neklesla pod 0 ani neprekročila $2N$. (Ak je aktuálna suma $\leq N$ a ešte mám mince, jednu mu dám, v opačnom prípade dá predavač jednu mincu mne.)

Predstavme si graf, ktorého vrcholy sú čísla od 0 do $2N$, hrana medzi i a j vedie práve vtedy, ak máme mincu hodnoty $|i-j|$. Ak teda máme K mincí, z každého vrcholu vychádza $O(K)$ hrán. Platenie mincami zodpovedá prechodom po hranách tohto grafu. Chceme teda nájsť v ňom najkratšie cesty z vrcholu 0 do každého z vrcholov 1 až N. To vieme spraviť prehľadávaním do šírky v čase $O(NK)$.

Jeden technický detail. Nebudeme minimalizovať priemer takto získaných hodnôt, ale ich súčet. Výsledok je ten istý, lebo priemer je súčet deleno N a N sa nemení. Získame tým to, že sa vyhneme použitiu „reálnych“ čísel.

[3] Nalej deci?

Listing programu:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

long long vyhodnot(int N, const vector<int> &mince) {
  vector<int> treba(2*N+1,0);
  vector<int> bol(2*N+1,0);
  queue<int> Q;
  bol[0]=1; Q.push(0);
  int C = mince.size();

  while (!Q.empty()) {
    int kde = Q.front(); Q.pop();
    for (int i=0; i<C; i++) {
      int kam;

      kam = kde + mince[i];
      if (kam > 2*N) continue;
      if (!bol[kam]) { bol[kam]=1; treba[kam]=treba[kde]+1; Q.push(kam); }

      kam = kde - mince[i];
      if (kam < 0) continue;
      if (!bol[kam]) { bol[kam]=1; treba[kam]=treba[kde]+1; Q.push(kam); }
    }
  }
  for (int i=1; i<=N; i++) if (!bol[i]) return 98765432198765432 1LL;
  long long res=0;
  for (int i=1; i<=N; i++) res += treba[i];
  return res;
}
```

Máme teda funkciu, ktorá nám vyhodnotí danú sadu mincí. (Všimnite si, že ak sa niektorá suma zaplatiť nedá, naša funkcia vráti veľké číslo, teda taká sada je veľmi zlá.) Nuž, čo s takouto funkciou?

Samozrejme, budeme generovať sady mincí a testovať, nakoľko sú dobré. A nebudeme to robiť len tak ledajako, ale prefíkane. Metóda, ktorú použijeme, sa v umelej inteligencii honosne nazýva *hill climbing*, teda lezenie na kopec.

Princíp je nasledovný: Máme množinu možných riešení a funkciu, ktorá nám hovorí, nakoľko je riešenie dobré.[4] Množinu riešení si predstavme ako body v teréne, pričom naša funkcia nám hovorí nadmorskú výšku daného bodu. My by sme chceli nájsť globálne maximum funkcie, najlepšie možné riešenie, teda najvyšší kopec v našej krajine. Toto je ale niekedy ťažké, až priam nemožné, preto sa uspokojíme s nájdením nejakého dosť vysokého kopca. Ako ho budeme hľadať? Postavíme sa na nejaké náhodné miesto, začneme liezť na nejaký z okolitých kopcov a lezieme, kým to ide. (Obľúbená finta navyše: ak máme viac možností, kam liezť, vyberieme si tú, ktorá je práve najstrmšia – ňou sa „za rovnaký čas“ dostaneme najvyššie.) Ak už sme na vrchole, pozrieme si jeho výšku (či sme náhodou neprekonali svoj osobný rekord) a náhodne sa presunieme niekam inam.

Všeobecná schéma takého hill climbing algoritmu je teda:

[4] V našom prípade máme funkciu, ktorá hovorí, nakoľko je riešenie zlé. Jedno znamienko mínus tento problém rieši.

```
globalneMaximum = -nekonecno;
aktualneRiesenie = nahodneRiesenie();

do zblbnutia opakuj: {
    lokalneMaximum = - nekonecno;
    aktualnaHodnota = ohodnot( aktualneRiesenie );

    forall (noveRiesenie podobné ako aktualneRiesenie) {
        toto = ohodnot( noveRiesenie );
        ak (toto > lokalneMaximum) {
            lokalneMaximum = toto;     dalsieRiesenie = noveRiesenie;
        }
    }

    ak (lokalneMaximum > aktualnaHodnota) {
        aktualneRiesenie = dalsieRiesenie;
        ak (lokalneMaximum > globalneMaximum) {
            globalneMaximum = lokalneMaximum;     vypis( aktualneRiesenie );
        }
    } inak aktualneRiesenie = nahodneRiesenie();
}
```

Ako preložiť metaforu o lezení na kopec na riešenie nášho problému? Bod, kde stojíme, je aktuálna sada mincí. Preskúmame nejaké podobné sady mincí a zistíme, či sú niektoré z nich od aktuálnej sady lepšie. Ak áno, vyberieme najlepšiu z nich a pokračujeme s ňou, ak nie, vygenerujeme si novú náhodnú sadu mincí a začneme odznova.

Zostáva zodpovedať dva posledné detaily: ako generovať začiatočnú sadu mincí a ktoré sady mincí skúšať ako podobné tej, ktorú práve máme.

Ja som v mojom programe novú sadu mincí generoval náhodne a následne utriedil, ako lokálne zmeny som skúšal zmenšiť hodnotu jednej mince (nanajvýš po najbližšiu menšiu), zväčšiť hodnotu jednej mince (až po najbližšiu väčšiu) a naraz zmeniť hodnoty max. 4 mincí o ± 1. Treba sa s tým rozumne pohrať, veľa možných lokálnych zmien síce znamená, že prezriete viac možností, ale toto prezeranie aj dlho trvá. Niekedy je lepšie skúšať lokálnych zmien menej, ale veľakrát.

Na záver uvádzame pre zaujímavosť najlepšie nájdené sady mincí a pre každého riešiteľa priemerné počty mincí pre ním navrhnuté sady. (Všimnite si, že keďže môj program nedostal týždeň strojového času, najlepšie nájdené riešenia pochádzajú od Miša Daniláka... a boli aj príslušne ohodnotené.)

$N=100$, $K=2$: 9, 10
$N=50$, $K=4$: 12, 13, 15, 22
$N=1000$, $K=12$: 3, 68, 95, 106, 118, 250, 367, 412, 414, 430, 445, 469
$N=10000$, $K=15$: 117, 220, 428, 1310, 1698, 1836, 2393, 2536, 2575, 2790, 2937, 2993, 3479, 4634, 4836

Mišo Danilák	6.7	2.54	2.995	3.8443
naše riešenie	6.7	2.54	3.012	3.8513
Peťo Perešíni	6.7	2.54	3.040	3.8835
Tomáš Pavlíček	6.7	2.54	3.074	3.9206
Matúš Fedák	6.7	2.54	3.026	3.9621
Martin Windisch	13.5	2.74	3.165	4.0911
Ondržo Bílka	7.5	3.02	3.373	4.1386
Mišo Kováč	7.5	2.94	3.333	4.3075
Martin Králik	7.3	2.9	3.184	11.3393
Anton Nagy	25.02	2.8	6.912	25.3495
Jano Mikuláš	6.7	2.54	—	—
Juro Ďuďák	11.36	3.52	—	—

Výsledková listina po 1. kole kategórie KSP

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
1	Perešíni Peter	Gym. Tajovského B. Bystrica	4	15	12	15	16	14	72
2	Bílka Ondřej	Gym. Zlín	4	15	15	13	15	7	65
3	Danilák Michal	Gym. Hubeného BA	3	12	12	15	4	16	59
4	Fedák Matúš	Gym. Stará Ľubovňa	4	12	10	15	8	13	58
5	Bundala Daniel	Gym. Jura Hronca BA	4	12	15	15	15		57
5	Imriška Jakub	Gym. Jura Hronca BA	4	12	15	15	15		57
7	Králik Martin	Gym. Grösslingová BA	4	12	10	15	4	4	45
7	Windisch Martin	Gym. Vrútky	4	8	9	9	8	11	45
9	Herman Peter	Gym. Jura Hronca BA	3	13	12	11	8		44
10	Mikuláš Ján	Gym. Haličská Lučenec	4	10	10	15		5	40
11	Rampášek Ladislav	Gym. Jura Hronca BA	3	12	12	15			39
12	Pavlíček Tomáš	SPŠE Piešťany	3	8	10	2	4	13	37
13	Ďuďák Juraj	Gym. Golianova Nitra	4	12	10	7	4	2	35
13	Mikuláš Ondrej	Gym. Haličská Lučenec	3	12	10	10	3		35
15	Petruchová Zuzana	Gym. Grösslingová BA	4	13	10	10			33
16	Jerguš Ján	Gym. Alejová Košice	3	13	10		8		31
17	Brezáni Samuel	Gym. Rajec	3	8	8	7	7		30
18	Danko Juraj	Gym. Jura Hronca BA	3	14	10	4			28
19	Kováč Michal	Gym. Grösslingová BA	4	1	9	7	0	9	26
20	Tomcsányi György	Gym. H. Selyeho Komárno	3	8	10		7		25
21	Pančík Andrej	Gym. Tajovského B. Bystrica	3	8	8		7		23
22	Halamíček Radovan	Gym. Jura Hronca BA	4	12	9				21
23	Tříska Martin	Gym. P. de Coubertina Piešťany	3	8	9	3			20
24	Nagy Anton	Gym. maďarské BA	1	1	9	2	0	5	17
25	Sudolský Michal	Gym. Tajovského B. Bystrica	3	12					12
26	Bašista Peter	Gym. P. Horova Michalovce	4	8					8