Important: Before reading GB_GAMES, please read or at least skim the programs for GB_GRAPH and GB_IO.

1. Introduction. This GraphBase module contains the *games* subroutine, which creates a family of undirected graphs based on college football scores. An example of the use of this procedure can be found in the demo program FOOTBALL.

⟨ gb_games.h 1 ⟩ ≡
```
extern Graph *games( );
```
See also section 5.

2. The subroutine call *games*(*n*, *ap0_weight*, *upi0_weight*, *ap1_weight*, *upi1_weight*, *first_day*, *last_day*, *seed*) constructs a graph based on the information in `games.dat`. Each vertex of the graph corresponds to one of 120 football teams at American colleges and universities (more precisely, to the 106 college football teams of division I-A together with the 14 division I-AA teams of the Ivy League and the Patriot League). Each edge of the graph corresponds to one of the 638 games played between those teams during the 1990 season.

An arc from vertex u to vertex v is assigned a length representing the number of points scored by u when playing v. Thus the graph isn't really "undirected," although it is true that its arcs are paired (i.e., that u played v if and only if v played u). A truly undirected graph with the same vertices and edges can be obtained by applying the *complement* routine of GB_BASIC.

The constructed graph will have $\min(n, 120)$ vertices. If n is less than 120, the n teams will be selected by assigning a weight to each team and choosing the n with largest weight, using random numbers to break ties in case of equal weights. Weights are computed by the formula

$$ap0\_weight \cdot ap0 + upi0\_weight \cdot upi0 + ap1\_weight \cdot ap1 + upi1\_weight \cdot upi1\,,$$

where *ap0* and *upi0* are the point scores given to a team in the Associated Press and United Press International polls at the beginning of the season, and *ap1* and *upi1* are the similar scores given at the end of the season. (The *ap* scores were obtained by asking 60 sportswriters to choose and rank the top 25 teams, assigning 25 points to a team ranked 1st and 1 point to a team ranked 25th; thus the total of each of the *ap* scores, summed over all teams, is 19500. The *upi* scores were obtained by asking football coaches to choose and rank the top 15 teams, assigning 15 points to a team ranked 1st and 1 point to a team ranked 15th. In the case of *upi0*, there were 48 coaches voting, making 5760 points altogether; but in the case of *upi1*, 59 coaches were polled, yielding a total of 7080 points. The coaches agreed not to vote for any team that was on probation for violating NCAA rules, but the sportswriters had no such policy.)

Parameters *first_day* and *last_day* can be used to vary the number of edges; only games played between *first_day* and *last_day*, inclusive, will be included in the constructed graph. Day 0 was August 26, 1990, when Colorado and Tennessee competed in the Disneyland Pigskin Classic. Day 128 was January 1, 1991, when the final end-of-season bowl games were played. About half of each team's games were played between day 0 and day 50. If *last_day* = 0, the value of *last_day* is automatically increased to 128.

As usual in GraphBase routines, you can set $n = 0$ to get the default situation where n has its maximum value. For example, either *games*$(0,0,0,0,0,0,0,0)$ or *games*$(120,0,0,0,0,0,0,0)$ produces the full graph; *games*$(0,0,0,0,0,50,0,0)$ or *games*$(120,0,0,0,0,50,0,0)$ or *games*$(120,0,0,0,0,50,128,0)$ produces the graph for the last half of the season. One way to select a subgraph containing the 30 "best" teams is to ask for *games*$(30,0,0,1,2,0,0,0)$, which adds the votes of the sportswriters to the votes of the coaches (considering that a coach's first choice is worth 30 points while a sportswriter's first choice is worth only 25). It turns out that 67 of the teams did not receive votes in any of the four polls; the subroutine call *games*$(53,1,1,1,1,0,0,0)$ will pick out the 53 teams that were selected at least once by some sportswriter or coach, and *games*$(67,-1,-1,-1,-1,0,0,0)$ will pick out the 67 that were not. A random selection of 60 teams can be obtained by calling *games*$(60,0,0,0,0,0,0,s)$. Different choices of the seed number s will produce different selections in a system-independent manner; any value of s between 0 and $2^{31}-1$ is permissible. If you ask for *games*$(120,0,0,0,0,0,0,s)$ with different choices of s, you always get the full graph, but the vertices will appear in different (random) orderings depending on s.

Parameters *ap0_weight*, *upi0_weight*, *ap1_weight*, and *upi1_weight* must be at most $2^{17} = 131072$ in absolute value.

```
#define MAX_N 120
#define MAX_DAY 128
#define MAX_WEIGHT 131072
#define ap u.I    /* Associated Press scores: (ap0 ≪ 16) + ap1 */
#define upi v.I   /* United Press International scores (upi0 ≪ 16) + upi1 */
```

3. Most of the teams belong to a "conference," and they play against almost every other team that belongs to the same conference. For example, Stanford and nine other teams belong to the Pacific Ten conference. Eight of Stanford's eleven games were against other teams of the Pacific Ten; the other three were played against Colorado (from the Big Eight), San José State (from the Big West) and Notre Dame (which is independent). The graphs produced by *games* therefore illustrate "cliquey" patterns of social interaction.

Eleven different conferences are included in `games.dat`. Utility field $z.S$ of a vertex is set to the name of a team's conference, or to Λ if that team is independent. (Exactly 24 of the I-A football teams were independent in 1990.) Two teams u and v belong to the same conference if and only if $u\neg conference \equiv v\neg conference$ and $u\neg conference \neq \Lambda$.

```
#define conference z.S
```

4. Each team has a nickname, which is recorded in utility field $y.S$. For example, Georgia Tech's team is called the Yellow Jackets. Six teams (Auburn, Clemson, Memphis State, Missouri, Pacific, and Princeton) are called the Tigers, and five teams (Fresno State, Georgia, Louisiana Tech, Mississippi State, Yale) are called the Bulldogs. But most of the teams have a unique nickname, and 94 distinct nicknames exist.

A shorthand code for team names is also provided, in the *abbr* field.

```
#define nickname y.S
#define abbr x.S
```

5. If a points to an arc from u to v, utility field $a\neg a.I$ contains the value 3 if u was the home team, 1 if v was the home team, and 2 if both teams played on neutral territory. The date of that game, represented as a integer number of days after August 26, 1990, appears in utility field $a\neg b.I$. The arcs in each vertex list $v\neg arcs$ appear in reverse order of their dates: last game first and first game last.

```
#define HOME 1
#define NEUTRAL 2    /* this value is halfway between HOME and AWAY */
#define AWAY 3
#define venue a.I
#define date b.I
```

⟨ gb_games.h 1 ⟩ +≡

```
#define ap u.I    /* repeat the definitions in the header file */
#define upi v.I
#define abbr x.S
#define nickname y.S
#define conference z.S
#define HOME 1
#define NEUTRAL 2
#define AWAY 3
#define venue a.I
#define date b.I
```

6. If the *games* routine encounters a problem, it returns Λ (`NULL`), after putting a code number into the external variable *panic_code*. This code number identifies the type of failure. Otherwise *games* returns a pointer to the newly created graph, which will be represented with the data structures explained in GB_GRAPH. (The external variable *panic_code* is itself defined in GB_GRAPH.)

```
#define panic(c) { panic_code = c; gb_trouble_code = 0; return Λ; }
```

7. The C file `gb_games.c` has the following overall shape:

```
#include "gb_io.h"      /* we will use the GB_IO routines for input */
#include "gb_flip.h"    /* we will use the GB_FLIP routines for random numbers */
#include "gb_graph.h"   /* we will use the GB_GRAPH data structures */
#include "gb_sort.h"    /* and gb_linksort for sorting */
⟨ Preprocessor definitions ⟩
⟨ Type declarations 11 ⟩
⟨ Private variables 13 ⟩
⟨ Private functions 23 ⟩
Graph *games(n, ap0_weight, upi0_weight, ap1_weight, upi1_weight, first_day, last_day, seed)
    unsigned long n;     /* number of vertices desired */
    long ap0_weight;     /* coefficient of ap0 in the weight function */
    long ap1_weight;     /* coefficient of ap1 in the weight function */
    long upi0_weight;    /* coefficient of upi0 in the weight function */
    long upi1_weight;    /* coefficient of upi1 in the weight function */
    long first_day;      /* lower cutoff for games to be considered */
    long last_day;       /* upper cutoff for games to be considered */
    long seed;           /* random number seed */
{ ⟨ Local variables 8 ⟩
  gb_init_rand(seed);
  ⟨ Check that the parameters are valid 9 ⟩;
  ⟨ Set up a graph with n vertices 10 ⟩;
  ⟨ Read the first part of games.dat and compute team weights 14 ⟩;
  ⟨ Determine the n teams to use in the graph 19 ⟩;
  ⟨ Put the appropriate edges into the graph 21 ⟩;
  if (gb_close( ) ≠ 0) panic(late_data_fault);
      /* something's wrong with "games.dat"; see io_errors */
  gb_free(working_storage);
  if (gb_trouble_code) {
    gb_recycle(new_graph);
    panic(alloc_fault);     /* oops, we ran out of memory somewhere back there */
  }
  return new_graph;
}
```

8. ⟨ Local variables 8 ⟩ ≡

```
Graph *new_graph;     /* the graph constructed by games */
register long j, k;   /* all-purpose indices */
```

This code is used in section 7.

9. ⟨ Check that the parameters are valid 9 ⟩ ≡

```
if (n ≡ 0 ∨ n > MAX_N) n = MAX_N;
if (ap0_weight > MAX_WEIGHT ∨ ap0_weight < −MAX_WEIGHT ∨ upi0_weight > MAX_WEIGHT ∨ upi0_weight <
      −MAX_WEIGHT ∨ ap1_weight > MAX_WEIGHT ∨ ap1_weight < −MAX_WEIGHT ∨ upi1_weight >
      MAX_WEIGHT ∨ upi1_weight < −MAX_WEIGHT) panic(bad_specs);
    /* the magnitude of at least one weight is too big */
if (first_day < 0) first_day = 0;
if (last_day ≡ 0 ∨ last_day > MAX_DAY) last_day = MAX_DAY;
```

This code is used in section 7.

10. ⟨ Set up a graph with n vertices 10 ⟩ ≡

```
new_graph = gb_new_graph(n);
if (new_graph ≡ Λ) panic(no_room);      /* out of memory before we're even started */
sprintf(new_graph→id, "games(%lu,%ld,%ld,%ld,%ld,%ld,%ld,%ld)", n, ap0_weight, upi0_weight,
    ap1_weight, upi1_weight, first_day, last_day, seed);
strcpy(new_graph→util_types, "IIZSSSIIZZZZZZ");
```

This code is used in section 7.

11. Vertices. As we read in the data, we construct a list of nodes, each of which contains a team's name, nickname, conference, and weight. After this list has been sorted by weight, the top n entries will be the vertices of the new graph.

⟨ Type declarations 11 ⟩ ≡

```
typedef struct node_struct {    /* records to be sorted by gb_linksort */
  long key;    /* the nonnegative sort key (weight plus 2^30) */
  struct node_struct *link;    /* pointer to next record */
  char name[24];    /* "College Name" */
  char nick[22];    /* "Team Nickname" */
  char abb[6];    /* "ABBR" */
  long a0, u0, a1, u1;    /* team scores in press polls */
  char *conf;    /* pointer to conference name */
  struct node_struct *hash_link;    /* pointer to next ABBR in hash list */
  Vertex *vert;    /* vertex corresponding to this team */
} node;
```

This code is used in section 7.

12. The data in `games.dat` appears in two parts. The first 120 lines have the form

```
ABBR College Name(Team Nickname)Conference;a0,u0;a1,u1
```

and they give basic information about the teams. An internal abbreviation code `ABBR` is used to identify each team in the second part of the data.

The second part presents scores of the games, and it contains two kinds of lines. If the first character of a line is '`>`', it means "change the current date," and the remaining characters specify a date as a one-letter month code followed by the day of the month. Otherwise the line gives scores of a game, using the `ABBR` codes for two teams. The scores are separated by '`@`' if the second team was the home team and by '`,`' if both teams were on neutral territory.

For example, two games were played on December 8, namely the annual Army-Navy game and the California Raisin Bowl game. These are recorded in three lines of `games.dat` as follows:

```
>D8
NAVY20@ARMY30
SJSU48,CMICH24
```

We deduce that Navy played at Army's home stadium, losing 20 to 30; moreover, San José State played Central Michigan on neutral territory and won, 48 to 24. (The California Raisin Bowl is traditionally a playoff between the champions of the Big West and Mid-American conferences.)

13. In order to map ABBR codes to team names, we use a simple hash coding scheme. Two abbreviations with the same hash address are linked together via the *hash_link* address in their node.

The constants defined here are taken from the specific data in games.dat, because this routine is not intended to be perfectly general.

```
#define HASH_PRIME 1009
⟨ Private variables 13 ⟩ ≡
  static long ma0 = 1451, mu0 = 666, ma1 = 1475, mu1 = 847;
    /* maximum poll values in the data */
  static node *node_block;    /* array of nodes holding team info */
  static node **hash_block;    /* array of heads of hash code lists */
  static Area working_storage;    /* memory needed only while games is working */
  static char **conf_block;    /* array of conference names */
  static long m;    /* the number of conference names known so far */
```

This code is used in section 7.

14. ⟨ Read the first part of games.dat and compute team weights 14 ⟩ ≡

```
  node_block = gb_typed_alloc(MAX_N + 2, node, working_storage);    /* leave room for string overflow */
  hash_block = gb_typed_alloc(HASH_PRIME, node *, working_storage);
  conf_block = gb_typed_alloc(MAX_N, char *, working_storage);
  m = 0;
  if (gb_trouble_code) {
    gb_free(working_storage);
    panic(no_room + 1);    /* nowhere to copy the data */
  }
  if (gb_open("games.dat") ≠ 0) panic(early_data_fault);
      /* couldn't open "games.dat" using GraphBase conventions; io_errors tells why */
  for (k = 0; k < MAX_N; k++) ⟨ Read and store data for team k 15 ⟩;
```

This code is used in section 7.

15. ⟨ Read and store data for team k 15 ⟩ ≡

```
  { register node *p;
    register char *q;
    p = node_block + k;
    if (k) p→link = p − 1;
    q = gb_string(p→abb, ' ');
    if (q > &p→abb[6] ∨ gb_char() ≠ ' ') panic(syntax_error);    /* out of sync in games.dat */
    ⟨ Enter p→abb in the hash table 16 ⟩;
    q = gb_string(p→name, '(');
    if (q > &p→name[24] ∨ gb_char() ≠ '(') panic(syntax_error + 1);    /* team name too long */
    q = gb_string(p→nick, ')');
    if (q > &p→nick[22] ∨ gb_char() ≠ ')') panic(syntax_error + 2);    /* team nickname too long */
    ⟨ Read the conference name for p 17 ⟩;
    ⟨ Read the press poll scores for p and compute p→key 18 ⟩;
    gb_newline();
  }
```

This code is used in section 14.

16. ⟨ Enter $p\rightarrow abb$ in the hash table 16 ⟩ ≡

```
{ long h = 0;      /* the hash code */
  for (q = p->abb; *q; q++) h = (h + h + *q) % HASH_PRIME;
  p->hash_link = hash_block[h];
  hash_block[h] = p;
}
```

This code is used in section 15.

17. ⟨ Read the conference name for p 17 ⟩ ≡

```
{
  gb_string(str_buf, ';');
  if (gb_char() ≠ ';') panic(syntax_error + 3);      /* conference name clobbered */
  if (strcmp(str_buf, "Independent") ≠ 0) {
    for (j = 0; j < m; j++)
      if (strcmp(str_buf, conf_block[j]) ≡ 0) goto found;
    conf_block[m++] = gb_save_string(str_buf);
  found: p->conf = conf_block[j];
  }
}
```

This code is used in section 15.

18. The key value computed here will be between 0 and 2^{31}, because of the bound we've imposed on the weight parameters.

⟨ Read the press poll scores for p and compute $p\rightarrow key$ 18 ⟩ ≡

```
p->a0 = gb_number(10);
if (p->a0 > ma0 ∨ gb_char() ≠ ',') panic(syntax_error + 4);      /* first AP score clobbered */
p->u0 = gb_number(10);
if (p->u0 > mu0 ∨ gb_char() ≠ ';') panic(syntax_error + 5);      /* first UPI score clobbered */
p->a1 = gb_number(10);
if (p->a1 > ma1 ∨ gb_char() ≠ ',') panic(syntax_error + 6);      /* second AP score clobbered */
p->u1 = gb_number(10);
if (p->u1 > mu1 ∨ gb_char() ≠ '\n') panic(syntax_error + 7);      /* second UPI score clobbered */
p->key = ap0_weight*(p->a0)+upi0_weight*(p->u0)+ap1_weight*(p->a1)+upi1_weight*(p->u1)+#40000000;
```

This code is used in section 15.

19. Once all the nodes have been set up, we can use the *gb_linksort* routine to sort them into the desired order. It builds 128 lists from which the desired nodes are readily accessed in decreasing order of weight, using random numbers to break ties.

We set the abbreviation code to zero in every team that isn't chosen. Then games involving that team will be excluded when edges are generated below.

⟨ Determine the n teams to use in the graph 19 ⟩ ≡

```
{ register node *p;      /* the current node being considered */
  register Vertex *v = new_graph->vertices;      /* the next vertex to use */
  gb_linksort(node_block + MAX_N - 1);
  for (j = 127; j ≥ 0; j--)
    for (p = (node *) gb_sorted[j]; p; p = p->link) {
      if (v < new_graph->vertices + n) ⟨ Add team p to the graph 20 ⟩
      else p->abb[0] = '\0';      /* this team is not being used */
    }
}
```

This code is used in section 7.

20. ⟨ Add team p to the graph 20 ⟩ ≡

```
{
  v→ap = ((long)(p→a0) ≪ 16) + p→a1;
  v→upi = ((long)(p→u0) ≪ 16) + p→u1;
  v→abbr = gb_save_string(p→abb);
  v→nickname = gb_save_string(p→nick);
  v→conference = p→conf;
  v→name = gb_save_string(p→name);
  p→vert = v++;
}
```

This code is used in section 19.

21. Arcs. Finally, we read through the rest of `games.dat`, adding a pair of arcs for each game that belongs to the selected time interval and was played by two of the selected teams.

```
⟨ Put the appropriate edges into the graph 21 ⟩ ≡
  { register Vertex *u, *v;
    register long today = 0;      /* current day of play */
    long su, sv;      /* points scored by each team */
    long ven;      /* HOME if v is home team, NEUTRAL if on neutral ground */
    while (¬gb_eof( )) {
      if (gb_char( ) ≡ '>') ⟨ Change the current date 22 ⟩
      else gb_backup( );
      u = team_lookup( );
      su = gb_number(10);
      ven = gb_char( );
      if (ven ≡ '@') ven = HOME;
      else if (ven ≡ ',') ven = NEUTRAL;
      else panic(syntax_error + 8);      /* bad syntax in game score line */
      v = team_lookup( );
      sv = gb_number(10);
      if (gb_char( ) ≠ '\n') panic(syntax_error + 9);      /* bad syntax in game score line */
      if (u ≠ Λ ∧ v ≠ Λ ∧ today ≥ first_day ∧ today ≤ last_day) ⟨ Enter a new edge 24 ⟩;
      gb_newline( );
    }
  }
```

This code is used in section 7.

22.

```
⟨ Change the current date 22 ⟩ ≡
  { register char c = gb_char( );      /* month code */
    register long d;      /* day of football season */
    switch (c) {
    case 'A': d = −26; break;      /* August */
    case 'S': d = 5; break;      /* thirty days hath September */
    case 'O': d = 35; break;      /* October */
    case 'N': d = 66; break;      /* November */
    case 'D': d = 96; break;      /* December */
    case 'J': d = 127; break;      /* January */
    default: d = 1000;
    }
    d += gb_number(10);
    if (d < 0 ∨ d > MAX_DAY) panic(syntax_error − 1);      /* date was clobbered */
    today = d;
    gb_newline( );      /* now ready to read a non-date line */
  }
```

This code is used in section 21.

23. ⟨ Private functions 23 ⟩ ≡

```
static Vertex *team_lookup( )      /* read and decode an abbreviation */
{ register char *q = str_buf;      /* position in str_buf */
  register long h = 0;      /* hash code */
  register node *p;      /* position in hash list */
  while (gb_digit(10) < 0) {
    *q = gb_char( );
    h = (h + h + *q) % HASH_PRIME;
    q++;
  }
  gb_backup( );      /* prepare to re-scan the digit following the abbreviation */
  *q = '\0';      /* null-terminate the abbreviation just scanned */
  for (p = hash_block[h]; p; p = p→hash_link)
    if (strcmp(p→abb, str_buf) ≡ 0) return p→vert;
  return Λ;      /* not found */
}
```

This code is used in section 7.

24. We retain the convention of GB_GRAPH that the arc from v to u appears immediately after a matching arc from u to v when $u < v$.

⟨ Enter a new edge 24 ⟩ ≡

```
{ register Arc *a;
  if (u > v) { register Vertex *w;
    register long sw;
    w = u; u = v; v = w;
    sw = su; su = sv; sv = sw;
    ven = HOME + AWAY − ven;
  }
  gb_new_arc(u, v, su);
  gb_new_arc(v, u, sv);
  a = u→arcs;      /* a pointer to the new arc */
  if (v→arcs ≠ a + 1) panic(impossible + 9);      /* can't happen */
  a→venue = ven; (a + 1)→venue = HOME + AWAY − ven;
  a→date = (a + 1)→date = today;
}
```

This code is used in section 21.

25. Index. As usual, we close with an index that shows where the identifiers of *gb_games* are defined and used.

August 4, 2025 at 10:31

GB_GAMES

Preliminary work on the Stanford GraphBase project was supported in part by National Science Foundation grant CCR-86-10181.